jamk.fi

Ohjelmisto-option kehitys Aplicom A1-laitteelle

Case Fuel Alert

Matti Heiskanen

Opinnäytetyö
Syyskuu 2014

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Heiskanen, Matti	Opinnäytetyö	19.09.2014
	Sivumäärä 54	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Ohjelmisto-option kehitys Aplicom A1-laitteelle

Case Fuel Alert

Koulutusohjelma

Ohjelmistotekniikka

Työn ohjaaja(t)

Mieskolainen, Matti
Väänänen, Olli

Toimeksiantaja(t)

Aplicom Oy

Tiivistelmä

Opinnäytetyön tilaajana toimi Aplicom Oy. Työn tavoitteena oli toteuttaa Aplicom A1-telematiikkalaitteen ohjelmistoon erikseen ostettava lisäosa, eli ohjelmisto-optio, polttoainevarkauksien havaitsemiseen. Lisäksi tavoitteena oli selvittää Aplicomin ohjelmisto-option kehitysprosessi tuotekehityksen osalta.

Opinnäytetyön toteutus koostui kahdesta osasta, kehitysprosessin selvittämisestä vaihe vaiheelta, ja itse kehitystyöstä. Kehitystyö sisälsi ohjelmisto-option määrittelyn, suunnittelun, toteutuksen, testauksen ja dokumentoinnin.

Kehitettävän ohjelmisto-option tarkoitus oli mahdollistaa raskaiden ajoneuvojen polttoaineen määrän tarkkailu ja polttoainevarkauksien havaitseminen ja niistä hälyttäminen A1-telematiikkalaitteella.

Optiosta tehtiin kaksi versiota, joista ensimmäinen julkaistiin asiakaskäyttöön kevättalvella 2014 ja toinen kesällä 2014. Toiseen versioon tehtiin ensimmäisen version käyttöönottaneiden asiakkaiden toiveiden mukaan mahdollisuus tarkkailla kahta polttoainetankkia samanaikaisesti.

Työn tavoitteet saavutettiin täysimääräisesti. Kehitetty ohjelmisto-optio tuli osaksi Aplicomin tuotevalikoimaa ja sitä on toimitettu Aplicomin asiakkaille. Samalla myös Aplicomin ohjelmistoprosessin vaiheet saatiin selvitettyä.

Avainsanat (asiasanat)

ohjelmistotekniikka, telematiikka, Java, Aplicom, A1

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Heiskanen, Matti	Bachelor's thesis	19.09.2014
		Language of publication: Finnish
	Number of pages: 54	Permission for web publication: x

Title of publication

Development of a software option for Aplicom A1 device

Case Fuel Alert

Degree programme

Software Engineering

Tutor(s)

Mieskolainen, Matti

Väänänen, Olli

Assigned by

Aplicom Oy

Abstract

The thesis was assigned by Aplicom Oy. The purpose of the thesis was to develop a separately sold part (software option) for the software of Aplicom A1 telematics device for detecting fuel thefts. An additional objective for the thesis was to examine the software development process at Aplicom.

The thesis consists of two main parts, to disclosure of the phases of the development process and the development itself. The development work included the definition, design, implementation, testing and documentation of the software option.

The purpose of the option was to allow monitoring the fuel level of commercial vehicles and to alarm fuel thefts with the A1 device.

Two versions of the option were developed, the first of which was released in winter of 2014 and the second in summer of 2014. The second version was developed to meet the customers' needs to monitor two tanks at the same time.

The goals of the thesis were fulfilled completely. The developed software option became a part of Aplicom's product catalog and it has been delivered to Aplicom's customers. The study of the software process phases was also successful and productive.

Keywords/tags (subjects)

software engineering, telematics, Java, Aplicom, A1

Miscellaneous

Sisältö

Kuviot

Taulukot

Liitteet

SANASTO

AD(C)	Analog to Digital (Converter), elektroninen komponentti joka muuttaa analogisen jännitearvon tietokoneen ymmärtämäksi numeroksi
API	Application Programming Interface, ohjelmointirajapinta
AT	Attention command, standardoitu komento (GSM) modeemin ohjaamiseen
COP	Co-Processor, Aplicom A1-laitteen apuprosessori, joka hoitaa mm. reaaliaikaisuutta vaativat toiminnot laitteessa.
GLONASS	GLobalnaja NAvigatsionnaja Sputnikovaja Sistema, Venäjän puolustusministeriön hallinnoima vastine GPS:lle
GPRS	General Packet Radio Service, GSM-verkossa toimiva tiedonsiirtopalvelu
GPS	Global Positioning System, satelliitteihin perustuva maailmanlaajuinen ja Yhdysvaltain puolustusministeriön hallinnoima paikannusjärjestelmä
GSM	Global System for Mobile Communications, toisen sukupolven matkapuhelinverkon protokollat kuvaava standardi
JAD	Java Application Descriptor, tekstitiedosto, joka sisältää kuvauksen Java ME:llä tehdyn sovelluksen JAR-paketista
JAR	Java ARchive, zip-tiedostoformaattiin perustuva pakkaustiedosto Java-sovelluksen luokille ja muille resursseille
KVM	Kilo Virtual Machine, kevyt virtuaalikone Java-koodin ajamiseen sulautetuissa ympäristöissä
M2M	”Machine to machine”, laitteiden välinen (kommunikointi)
MES	Module Exchange Suite, Cinterionin moduuleiden ja PC:n väliseen tiedostojen siirtoon tarkoitettu ohjelmisto
OTAP	Over-The-Air-Provisioning, langattomasti tapahtuva ohjelmiston tai konfiguraation päivitys laitteelle
SMS	Short Message Service, GSM-verkossa toimiva tekstiviestipalvelu
SPI	Serial Peripheral Interface, sarjamuotoinen lisälaiterajapinta. Käytetään prosessorien ja niiden oheislaitteiden välisessä kommunikoinnissa.

1 TYÖN KUVAUS

Opinnäytetyön tilaajana toimi Aplicom Oy, joka on suomalainen telematiikka-, paikannus- ja viestintälaitteiden valmistaja. Sillä on toimipaikat Äänekoskella ja Espoossa. Äänekosken toimipaikalla on tuotekehitys, tuotanto ja tukipalvelut ja Espoossa myynti ja markkinointi. Aplicomin noin 30 työntekijästä suurin osa on Äänekoskella.

Aplicomin tuotteissa ohjelmistoilla on nykyään yhä suurempi merkitys, sillä itse laitteiston kehitys on suhteessa kallista ja kilpailu alalla on kovaa. Laitteiden ohjelmistokehityksessä on viime vuosina entistä enemmän panostettu sovelluksiin, jotka ovat hyvin laajasti asiakkaan konfiguroitavissa. Toinen tärkeä asia kehityksessä ovat erilaiset optiot asiakkaan valittaviksi tarpeen mukaan. Nykyisessä kovassa kilpailutilanteessa kustannustehokkuus korostuu, ja asiakkaat maksavat vain ominaisuuksista, joita he todella tarvitsevat.

Opinnäytetyön tärkeimpänä konkreettisena tavoitteena oli toteuttaa loppuasiakkaille käyttöön tuleva ohjelmisto-optio polttoaineen seurantaan. Lisäksi tavoitteena oli selvittää Aplicomin ohjelmisto-option kehitysprosessi tuotekehityksen osalta sekä samalla oppia A1-laitteen ohjelmiston arkkitehtuuri ja toiminta mahdollisimman hyvin.

Kehitysprosessin vaiheet selvitetään ideasta valmiiksi tuotteeksi sisältäen määrittelyn, suunnittelun, toteutuksen, testauksen ja tuotteistuksen. Opinnäytetyön toteutusosa sisältää ensimmäisen julkaistavan version ohjelmisto-optiosta ja option käyttöön ottaneilta asiakkailta tulleiden toiveiden mukaan kehitetyn toisen version.

2 TAUSTA JA LÄHTÖKOHDAT

2.1 Telematiikka

Telematiikka on (muun muassa) elektroniikkaa, langatonta tiedonsiirtoa, ajoneuvoteknologioita ja tietokonetekniikkaa yhdistelevä tieteenala. Telematiikka on hyvin laaja käsite, ja se voi sisältää tunnetuimpien sovellusten kuten paikannus- ja navigointijärjestelmien lisäksi esimerkiksi erilaisia antureita, älykortteja, Internet-tekniikkaa ja monenlaisen median välitystä. Usein telematiikasta puhuttaessa tarkoitetaan nimenomaan ajoneuvotelematiikkaa. Se on tietojen lähettämistä ja vastaanottamista langattomasti ajoneuvon ja esimerkiksi palvelimen tai satelliitin välillä. Yleisin esimerkki ajoneuvotelematiikasta on satelliitteihin perustuva GPS-paikannus. (Telematics 2014.)

Ajoneuvotelematiikalle on nykypäivänä paljon muitakin erilaisia sovelluksia, eikä kaikkia potentiaalisia käyttökohteita ole vielä tutkittu tai edes löydetty. Yksi suurimmista telematiikkaa hyödyntävistä aloista on nykyään entistä nopeatempoisempi, tiukemmin valvottu ja kovemmin kilpailtu logistiikka. On selvää, että rekkoja ja kuorma-autoja ei kannata ajattaa tyhjänä, vaan aina pitäisi olla myös paluumatkalle jokin kuorma. Kun toimistolla ajojärjestelijällä on tieto reaaliaikaisesti missä mikäkin auto liikkuu ja mikä sen tilanne on, voidaan ajot optimoida huomattavasti paremmin kuin vain etukäteen suunnittelemalla.

Pelkän paikkatiedon lisäksi paljon muutakin tietoa voidaan lähettää ilmateitse vaikka maapallon toiselle puolelle. Esimerkiksi Aplicomin A1-laitteen voi kytkeä todella monenlaisiin ulkoisiin laitteisiin, kuten vaikkapa nykyaikaisen kuorma-auton FMS-väylärajapintaan. Rajapinnasta saadaan monenlaisia tietoja moottorin kierroksista ja kaasupolkimen asennosta lähtien. (Fleet Management System 2014.)

Suuremman liikuteltavan massan vuoksi raskaalla kalustolla ajettaessa ajotavalla on selvästi suurempi vaikutus polttoaineenkulutukseen kuin henkilöautolla, ja kevyellä kaasujalalla ja ennakoinnilla säästää paljon. Kun kilometrejä voi tulla yhdelle autolle helposti

satatuhatta vuodessa, on telematiikan avulla ajotapaa seuraamalla ja sitä kautta kuljettajia ohjeistamalla mahdollisuus säästää huomattavia summia rahaa polttoainekuluissa.

Yksi merkittävä telematiikan tarjoama apu kuljetusyrityksille on lisäksi mahdollisuus ladata digitaalisten ajopiirturien tiedot suoraan autosta toimistolle, ilman että kuljettajien tarvitsee käydä lukemassa piirturikorttinsa joka kerta paikanpäällä.

Tällä tavalla telematiikkajärjestelmä voi maksaa itsensä nopeasti takaisin jo pelkästään polttoainesäästöillä, puhumattakaan työnohjauksen ja ajojärjestelyn reaaliaikaisuuden tuomista säästöistä.

2.2 Aplicom Oy

2.2.1 Historiaa

Aplicomin historia alkaa 1990-luvun alusta, silloisesta Nokia Mobiran Mobiledata-yksiköstä. Äänekoskella sijainnut yksikkö oli tehnyt tuotteita erilaisiin mobiilidata-verkkoihin, muun muassa ammattilaiskäyttöön suunnattuja radiopuhelimia. Sittemmin se oli erikoistunut datan välitykseen ja Ruotsissa alkunsa saaneen Mobitex-verkon laitteiden kehitykseen. (Savolainen 2014)

Nokian keskittäessä toimintaansa GSM:n ympärille tämä yksikkö siirtyi osaksi Computec Oy:tä Jyväskylään, ja tuotteiden kehitystä jatkettiin Aplicom tuotemerkillä. Tiivis yhteistyö Nokian kanssa jatkui, ja kun SMS-viestit tulivat käyttöön, voitiin Aplicom-tuotteilla liittyä esimerkiksi Nokian menestyksekkääseen 2210-puhelimeen. (Savolainen 2014)

Vuonna 1995 perustettiin Aplicom Oy, joka jatkoi itsenäisenä yrityksenä datansiirtotuotteiden kehitystä ja markkinointia. Toimipisteet olivat Äänekoskella ja Helsingissä, joista ensin mainitussa tehdas ja tuotekehitys ja jälkimmäisessä johto ja myynti. (Savolainen 2014)

Alkuun itsenäisen Aplicomin tuotteet olivat pääasiassa ajoneuvotietokoneita, joilla hoidettiin viestintää esimerkiksi kuljetusyrityksen toimiston ja autojen välillä ajojen seurannan, raportoinnin ja laskutuksen automatisoimiseksi. Suurelle osalle asiakkaista autojen

paikkatiedolla ei tuolloin ollut merkitystä, ja nykyisen kaltaista GPS-teknologiaa otettiin vasta vähitellen käyttöön. (Savolainen 2014)

Uusien asiakkaiden, kuten eri maiden autoliittojen tiepalvelujen, ambulanssien ja muiden paikkatietoa tarvitsevien sovellusalueiden myötä tarve GPS:lle tuli jatkuvasti suuremmaksi. Aluksi käytettiin ulkoisia GPS-vastaanottimia, mutta vähitellen ne alettiin sisällyttää tuotteisiin, ensin erikseen myytävinä optioina. Tuotteet suunniteltiin jo tuolloin kustannustehokkuuden ja joustavuuden takia mahdollisimman modulaarisiksi kokonaisuuksiksi, joista asiakas osti vain tarvitsemansa osat. Kuten nykyisistä tuotteista esimerkiksi A1 M2M, myös Aplicomin alkuaikoina tuotteita on toimitettu myös siten, että asiakas tai yhteistyökumppani kehittää itse ohjelmiston Aplicomin toimittamalle pohjalle. (Savolainen 2014)

Aplicomin tuotteet ovat aina olleet optimoituja vaativiin ajoneuvoympäristöihin. Tässä asiassa sulautettu lähestymistapa ja laitteen hallinta alimmalta tasolta lähtien on ollut tärkeää esimerkiksi käyttöjännitteen ongelmien hallintaan liittyen. Tätä hankittua osaamista hyödynnetään tänäkin päivänä, ja muun muassa A1- ja A9-laitteissa käytettävä kaksoisprosessoriarkkitehtuuri on tästä hyvä esimerkki. (Savolainen 2014)

2.2.2 Nykytilanne ja tulevaisuuden näkymät

Verrattuna Aplicomin alkuaikoihin, merkittävin muutos telematiikkalaitteiden kehityksessä on tapahtunut datan siirrossa laitteesta ulos. GSM-verkon GPRS-datansiirron kehitys on mahdollistanut standardoitujen tiedonsiirtoprotokollien, kuten TCP/IP ja UDP käyttöönoton myös mobiililaitteissa. GPRS on edelleen yleisin teknologia telematiikkalaitteissa, mutta nopeammat 3G- ja 4G-verkot ovat tulossa matkapuhelinpuolelta myös telematiikkaan. Ajoneuvoon asennettavilla laitteilla ja niiden käyttämillä teknologioilla on oleellinen rooli, kun telematiikassa halutaan hyödyntää ajoneuvojen omia väyliä tiedonsaantiin ja esimerkiksi etädiagnosointiin. Ajoneuvon omien väylien hyödyntämiseen liittyy myös kokonaan uusia palvelutarjonnan mahdollisuuksia, joihin liittyen Aplicom on aktiivinen tänä päivänä. (Savolainen 2014)

3 APLICOM A1

3.1 Yleistä

A1 on erityisesti vaativaan ammattikäyttöön suunnattu telematiikka- ja paikannuslaite. Sen mekaniikka on suunniteltu kestämään rankkoja olosuhteita, ja kahden prosessorin arkkitehtuurin takia sen toiminta on myös luotettavaa. A1:ssä on monipuoliset liitäntämahdollisuudet, joiden avulla se saadaan kytkettyä monenlaisiin ajoneuvon järjestelmiin ja lisälaitteisiin, kuten CAN-väylään, Garmin-navigaattoriin tai digitaaliseen ajopiirturiin. Kaikki sen liitännät ulkomaailmaan on myös suojattu muun muassa jännitepiikeiltä ja häiriöiltä, joita ajoneuvoympäristössä esiintyy. Myöskään kaapeleita väärin kytkemällä laitetta ei saa rikottua.

Tuoteperheeseen kuuluvat A1 MAX, A1 MAX RDL, A1 TRAX ja A1 M2M, joista MAX RDL:ssä on laajimmat ominaisuudet ja muissa hieman rajoitetummat ominaisuudet. Erot tuotteiden välillä tulevat pääasiassa ohjelmistoista, mutta pieniä eroja on myös HW-komponenteissa. Kaikista tuotteista on saatavilla myös itse ohjelmoitavissa oleva versio, jonka mukana toimitetaan tarvittavat työkalut ohjelmistokehitykseen. Tässä opinnäytetyössä tehtävä ohjelmisto-optio tulee käyttöön MAX-, MAX RDL- sekä TRAX-tuotteille. Kuviossa 1 on Aplicom A1 MAX.

Kuvio 1. Aplicom A1-laite

Valmiin ohjelmiston kanssa toimitettavien A1-laitteiden perusajatus on, että kaikki ominaisuudet ovat hyvin monipuolisesti käyttäjän konfiguroitavissa. Konfigurointi tarkoittaa tässä tapauksessa sitä, että sama laitteessa oleva ohjelmisto voidaan yhtä konfigurointitiedostoa muuttamalla säätää toimimaan lukemattomilla eri tavoilla. Tällä saavutetaan luonnollisesti huomattavasti joustavampi ja muunneltavampi tuote verrattuna siihen, että jokaiseen käyttötarkoitukseen olisi oma ohjelmistoversionsa.

Toinen tärkeä A1:n ominaisuus on laitteiden etäpäivitys. Kaikki laitteen ohjelmistot ja konfiguraatio voidaan päivittää etänä laitteen OTAP-ominaisuuden avulla. Päivitys voidaan tehdä myös asiakkaan toimesta. Etäpäivitys on luonnollisesti huomattavasti kätevämpi tapa laitteiden päivittämiseen kuin se, että esimerkiksi tuhat autoissa ympäri maailmaa olevaa laitetta pitäisi käydä manuaalisesti yksi kerrallaan päivittämässä.

3.2 Laitteen rakenne

3.2.1 Kahden prosessorin arkkitehtuuri

Verrattuna vanhempiin Aplicomin tuotteisiin, A1-laitteen kehitykselle on ollut vaatimuksena yhä nopeampi ja joustavampi ohjelmointi, jotta kilpailuun ja asiakastarpeisiin voidaan vastata nopeasti. Javalla ohjelmoitavat valmiit moduulit, jotka sisältävät myös GSM-osan ovat tarjonneet tähän hyvän ratkaisun. Java Micro Editionin (Java ME) sisältämät standarditoteutukset muun muassa kommunikointiin ja tietoturvaan helpottavat ja nopeuttavat kehitystä, kun kaikkea ei tarvitse tehdä itse tai ostaa erikseen. Javalla ohjelmoitavan moduulin vahvuus on siis sen mahdollistama helposti ylläpidettävän asiakassovelluksen nopea kehittäminen.

Telematiikkalaitteelle olennaisen tärkeät reaaliaikaiset toiminnot ja kyky toimia vaativassa ajoneuvoympäristössä on puolestaan järjestetty erillisellä prosessorilla, joka keskittyy hoitamaan ongelmallisen ympäristön haasteet. Erillisellä sulautetulla mikro-ohjaimella ja sen reaaliaikaisella moniajokäyttöjärjestelmällä saadaan hallittua esimerkiksi erilaiset väylät, pulssilaskurit ja muut nopeaa reaaliaikaisuutta vaativat toiminnot huomattavasti

sujuvammin. Sillä aikaa Java-puoli hoitaa asiakassovellusta ja kommunikointia taustajärjestelmän kanssa.

Mikro-ohjaimella saadaan myös erilaisia GPS:ään tai kiihtyvyysanturiin liittyviä arvoja analysoimalla tuotettua ilmoituksia Java-puolelle esimerkiksi kuljetusta matkasta tai kiihdytyksistä ja jarrutuksista.

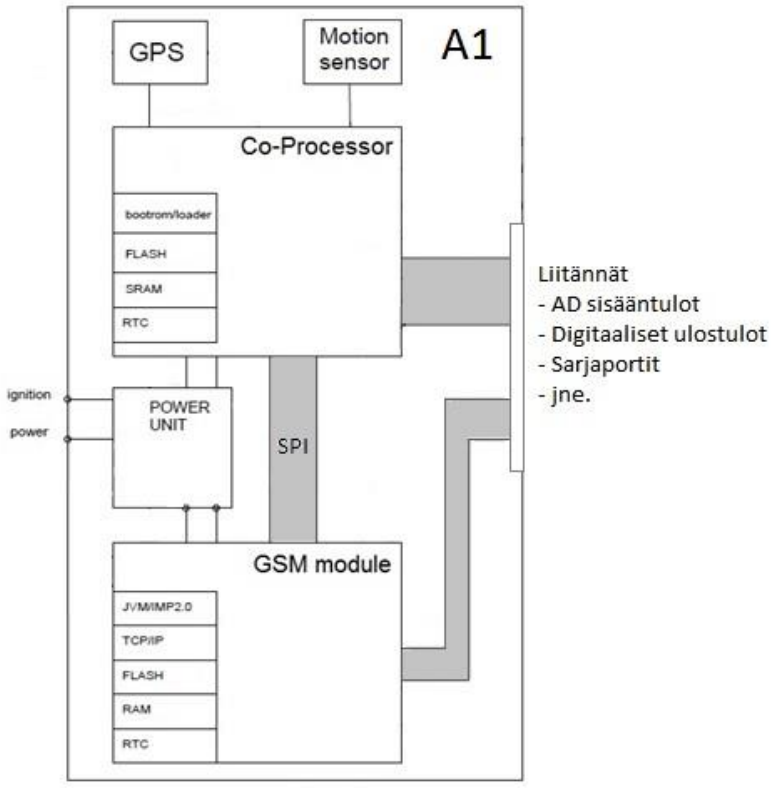
Prosessorit myös tarkkailevat toistensa toimintaa, ja jos jompikumpi huomaa että toinen ei toimi normaalisti, se voi käynnistää toisen uudestaan. Tämä lisää huomattavasti toimintavarmuutta kentällä.

3.2.2 GSM-moduuli

GSM-moduulina A1:ssä käytetään hollantilaisen Gemalton valmistamaa Cinterion TC65i-X-moduulia. Niin sanottu bisneslogiikka ja kaikki käyttäjälle suoraan näkyvät toiminnot tapahtuvat GSM-moduulin sisällä. Se hoitaa myös langattoman tiedonsiirron matkapuhelinverkon yli sekä SMS- että GPRS-muodossa. Moduulin toimintaa voidaan ohjata monipuolisesti tätä tarkoitusta varten olevilla AT-komennoilla. Komentojen antamista varten on useita AT-rajapintoja joihin päästään käsiksi ulkoisesti fyysisten porttien kautta tai ohjelmallisesti Java APIn kautta.

3.2.3 Co-Processor, eli apuprosessori

Toinen A1:n prosessoreista, niin sanottu apuprosessori (Co-Processor, COP) on Atmelin valmistama 32-bittiseen ARM7-ytimeen perustuva mikro-ohjain. Siinä pyörii moniajoon kykenevä OS95a-reaaliaikakäyttöjärjestelmä, ja sen vastuulla on reaaliaikainen käsittely mm. GPS:ltä, kiihtyvyysanturilta ja analogisten sisääntulojen AD-muuntimilta tulevalle datalle. Se myös ohjaa ulostuloja ja hoitaa kommunikoinnin A1:een liitettyjen laitteiden ja väylien, kuten CAN- ja 1-wire -liitäntöjen kanssa. Apuprosessorin ja GSM-moduulin välinen kommunikointi tapahtuu niiden välillä olevan SPI-väylän kautta. Kuviossa 2 on laitteen karkeahko lohkokaavio.



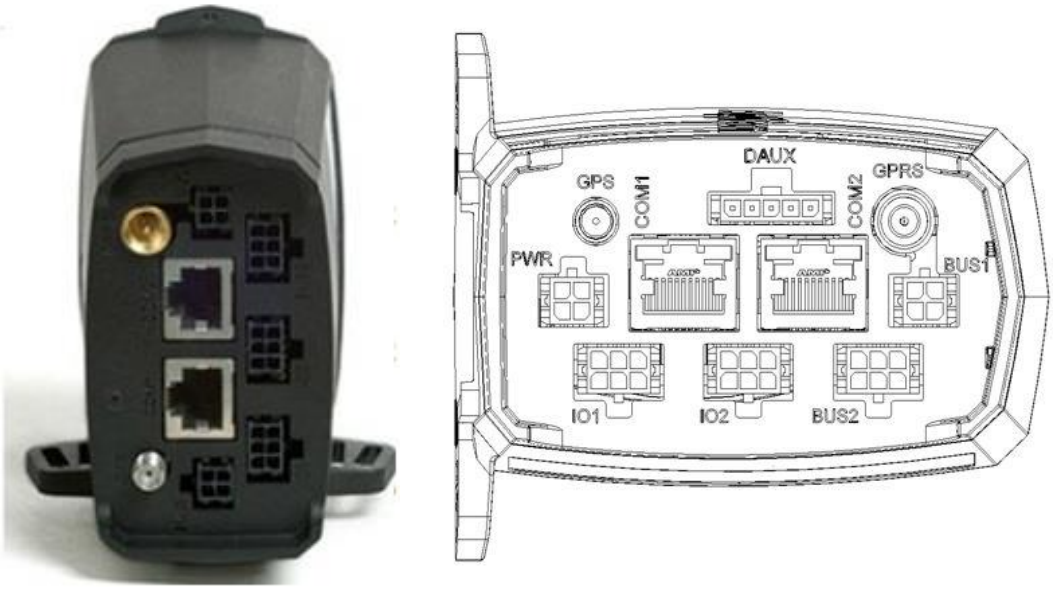
Kuvio 2. Aplicom A1-laitteen lohkokaavio

Nimestään huolimatta apuprosessori hoitaa niin huomattavan määrän datalle tehtävästä laskennasta ja sen tulkinnasta ennen sen toimittamista Java-puolelle, että ei voida ajatella sen olevan millään tavalla vähemmän tärkeä osa A1-laitetta kuin GSM-moduuli.

3.2.4 **Liitännät**

A1:ssä on monipuoliset liitännät ulkomaailmaan. Näihin sisältyy kaksi kappaletta digitaalisia ulostuloja ja kuusi kappaletta digitaalisia sisääntuloja, jotka voidaan vaihtoehtoisesti konfiguroida neljäksi analogiseksi sisääntuloksi ja kahdeksi pulssilaskuriksi. Analogisissa sisääntuloissa on 10-bittiset AD-muuntimet.

Lisäksi A1:ssä on kaksi RS232-sarjaporttia, yksi RS485-sarjaportti, K-linja-liitäntä, CAN-liitäntä ja 1-wire-liitäntä. Kaikki liitännät eivät tosin ole yhtä aikaa käytettävissä, sillä osa niistä käyttää samaa sisäistä porttia. Kuviossa 4 näkyvät kaikki laitteen ulkoiset liittimet.



Kuvio 3. A1-laitteen liittimet (A1 Installation Guide 2014)

3.3 Ohjelmistot

3.3.1 Apuprosessorin ohjelmisto

Co-Processor Software (COPSW), on A1-laitteen apuprosessorin ohjelmisto. Se on C-kielellä kirjoitettu ja konekielelle käännetty ohjelmisto, jonka ajamisesta mikro-ohjaimessa huolehtii OS95a-reaaliaikakäyttöjärjestelmä. COPSW hoitaa kaikki aikakriittiset asiat A1-laitteessa, kuten väylien ja I/O:n kanssa toiminnan. Se kommunikoi myös kiihtyvyysanturin ja GPS-moduulin kanssa, ja tulkitsee niiltä vastaanottamaansa dataa. COPSW:ssä on hyvin kehittyneet algoritmit GPS- ja kiihtyvyysanturidatan analysointiin, ja se osaa tulkita niistä luotettavasti monenlaisia asioita esimerkiksi ajotavasta. COPSW kommunikoi Java-puolen ohjelmiston kanssa SPI-väylän kautta.

3.3.2 GSM-moduulin ohjelmisto

A1:n GSM-moduulin Java-ohjelmisto perustuu Sun Microsystemsin kehittämään Java Micro Edition -sovellusympäristöön. Java Micro Edition (Java ME) on rajoittuneen suorituskyvyn omaaviin laitteisiin tarkoitettu ympäristö, ja verrattuna esimerkiksi PC-ympäristössä käytettävään Java Standard Editioniin, siinä on suppeammat luokkakirjastot ja se on muutenkin optimoitu yksinkertaisemmille laitteille, muun muassa näppäimistön käsittelyn osalta. (Java ME 2014.)

Javan ajatus on olla alustariippumaton ohjelmointiympäristö. Tämä saavutetaan sillä, että Java-ohjelmointikielellä tehdystä lähdekoodista käännetty tavukoodi ajetaan virtuaalikoneessa.

A1:n GSM-moduulissa on Sun Microsystemsin kehittämä KVM-virtuaalikone, jossa Java-tavukoodi ajetaan. KVM eli Kilo Virtual Machine on mobiililaitteisiin tarkoitettu huomattavasti kevyempi vastine tavalliselle PC-ympäristön Java Virtual Machinelle. Itse virtuaalikoneen koko on alle sata kilotavua. KVM-virtuaalikoneen ajaminen käyttää myös dynaamista muistia säästeliäämmin, vain joitain kymmeniä kilotavuja, ja näin ollen virtuaalikoneen laitteeseen implementointiin riittää yhteensä jopa vain 128 kilotavua muistia. (The K Virtual Machine 2014.)

Tavukoodi muistuttaa konekieltä, mutta on abstraktimpaa ja siten laitteistoriippumatonta. Tavukoodi koostuu käskyistä, joita virtuaalikone suorittaa. Tällaisia käskyjä ovat esimerkiksi ”iadd”, joka laskee yhteen kaksi kokonaislukua, tai ”fastore”, joka tallentaa float-liukuluvun taulukkoon. (Java Virtual Machine Specification 2013.)

KVM:ssä, kuten useimmissa nykyaikaisissa virtuaalikoneissa on myös Just-In-Time Compiler (JIT Compiler), joka kääntää käskyt suoraan konekielelle ajon aikana. JIT Compilerilla saavutetaan nopeampi koodin suoritus, mutta se luonnollisesti varaa toimintaansa myös muistia.

Konfiguraatiot

Java ME:stä on olemassa kaksi eri konfiguraatiota CDC (Connected Device Configuration) ja CLDC (Connected Limited Device Configuration). Ne määrittelevät käytössä olevat Javan ydinluokat ja virtuaalikoneen ominaisuudet. Konfiguraatioista CDC on tarkoitettu hieman PC-konetta vähemmät resurssit omaaville laitteille, ja CLDC nimensä mukaisesti vielä tästä rajoittuneemmille laitteille. (Connected Limited Device Configuration 2014.) A1-laitteen GSM-moduulissa käytetään CLDC 1.1-konfiguraatiota, jossa on versioon 1.0 verrattuna hieman enemmän ominaisuuksia, kuten mahdollisuus käyttää liukulukuja.

Profiilit

Konfiguraatioita täydentämään on olemassa profiileja, jotka mahdollistavat tietynlaisten laitteiden ohjelmoinnin niin, että sama ohjelma toimii kaikissa saman profiilin laitteissa. Profiileista tunnetuin on MIDP, eli Mobile Information Device Profile. Se on mobiililaitteiden ohjelmointiin tarkoitettu profiili, joka takaa että esimerkiksi profiilin mukaisesti tehty peli toimii kaikissa profiilin mukaisissa puhelimissa. (Java Platform, Micro Edition 2014.)

Information Module Profile (IMP) on tarkoitettu laitteille, joissa ei ole näyttöä ja joiden verkko-ominaisuudet ovat hieman rajallisemmat verrattuna MIDP-profiilin laitteisiin. IMP on osajoukko MIDP-profiilista, eli se sisältää osan MIDP:n ominaisuuksista, mutta

esimerkiksi graafisen käyttöliittymän ohjelmointi on jätetty pois. (Java Platform, Micro Edition 2014.)

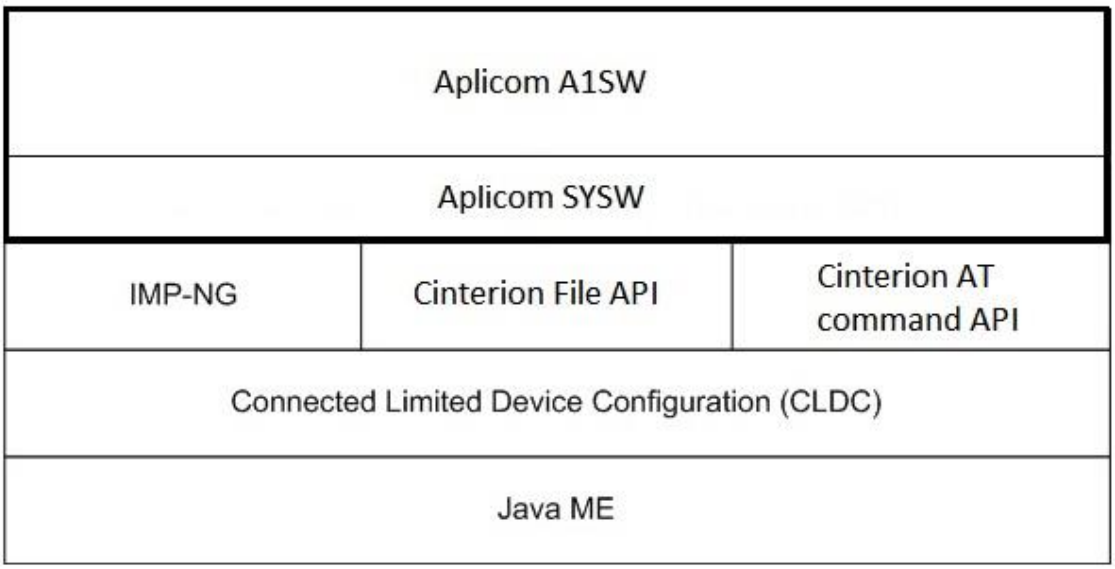
A1-laitteen GSM-moduulissa on käytössä IMP-NG (Next Generation), joka perustuu MIDP 2.0 versioon. Se sisältää aikaisempaan versioon nähden uudistetut turvallisuus- ja verkkotyypit ja -rajapinnat. (Java Platform, Micro Edition 2014.)

MIDP-profiilin alaisiin laitteisiin tehtyjä Java-sovelluksia kutsutaan MIDleteiksi, ja koska A1:ssä oleva IMP-NG perustuu myös MIDP-profiiliin, on myös A1-laitteen Java-sovellus MIDlet. (Java User's Guide 2012.)

Lisäksi moduulin ohjelmistoon kuuluvat Cinterionin tuottamat API:t tiedostojen käsittelyyn ja AT-komentojen käyttämiseen moduulin ohjauksessa.

Aplicomin Java-sovellus

A1:n Java-sovelluksessa on kaksi osaa, SYSW ja A1SW. SYSW on A1-laitteen Java-sovelluksen alempi kerros, jonka päälle A1SW rakentuu. Se sisältää laitteen käyttämiseen tarvittavat perustoiminnot, kuten kommunikoinnin apuprosessorin kanssa ja yhteyden Cinterionin API:hin. Asiakkaat jotka ostavat Aplicomilta vain laitteen ja A1 SDK:n ja kehittävät itse ohjelmiston laitteeseen A1SW:n tilalle käyttävät SYSW:tä ohjelmistojensa pohjana. Kuviossa 4 on kuvattu miten A1:n Java-sovellus rakentuu Java ME:n, konfiguraation, profiilin ja moduulivalmistajan API:en päälle.

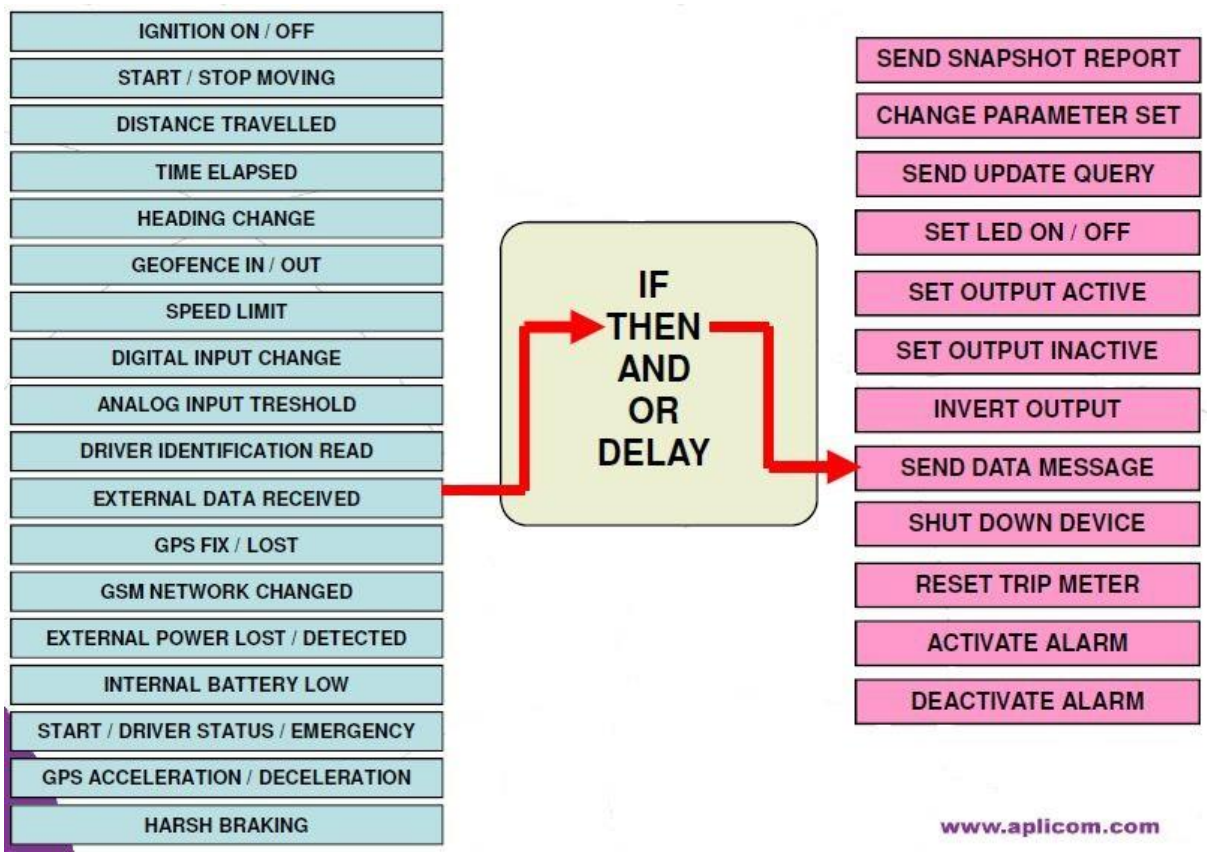


Kuvio 4. GSM-moduulin Java-ohjelmisto

A1SW

A1SW on laitteen varsinainen äly. Se on hyvin laajasti konfiguroitavissa, ja sen takia se soveltuu todella moneen eri käyttöön. Yksinkertaistettuna A1SW on niin sanottu tapahtuma-toiminto-tilakone, eli se odottaa tapahtumia ja suorittaa toimintoja niiden mukaan ja ne suoritettuaan palaa taas odottamaan uusia tapahtumia.

Esimerkiksi, jos A1SW havaitsee, että lähdettiin liikkeelle, se voi lähettää palvelimelle viestin. Tapahtumien ja toimintojen välille voi luoda monenlaisia ja monimutkaisiakin yhteyksiä ja erilaisia ehtoja tai viiveitä toimintojen suorittamiseen. Kuviossa 5 on esimerkkejä mahdollisista tapahtumista joihin A1-laite voi reagoida ja toiminnoista joita se voi suorittaa valitulla tavalla.



Kuvio 5. Esimerkkejä A1-laitteen tapahtumista ja toiminnoista. (Aplicom A1 with ready made software 2009)

Tapahtumat ja niitä vastaavat toiminnot ovat täysin käyttäjän konfiguroitavissa, ja erilaisia variaatioita konfiguraatiosta voi olla äärettömästi. Konfigurointi toimii niin, että xml-muotoinen konfigurointitiedosto ladataan laitteen muistiin, josta Java-sovellus lataa sen käyttöönsä käynnistyksen yhteydessä. Sen perusteella asetellaan muun muassa kommunikointiin, I/O:hon ja kytkettyihin lisälaitteisiin liittyviä asetuksia. Lisäksi konfiguraatiosta A1SW tietää, miten sen tulee reagoida mihinkin tapahtumiin, kuten esimerkiksi mitä tehdään, jos nopeus ylittää tietyn rajan tai jonkin I/O-linjan tila muuttuu.

Kommunikointi

A1SW:ssä on hyvin kehittynyt ja monipuolinen järjestelmä kommunikoinnin hallintaan. Se tukee GPRS- ja SMS-muotoista tiedonsiirtoa, ja niiden kautta voidaan välittää usean eri protokollan mukaisesti muotoiltuja viestejä.

Myös ongelmatilanteet verkon kuuluvuuden kanssa on otettu huomioon. Esimerkiksi roaming-tila voidaan tunnistaa ja siten kommunikoinnin kustannuksia voidaan hallita rajoittamalla viestin lähetystä ulkomailla. Viestit voidaan myös priorisoida esimerkiksi siten, että vain tärkeimmät viestit, kuten tieto liikkeellelähdöstä ja pysähtymisestä lähetetään kotiverkon ulkopuolella oltaessa, ja muut vähemmän tärkeät viestit vasta palattaessa kotiverkon alueelle. A1SW myös tallentaa viestit laitteen flash-muistiin, jos niitä ei saada lähetettyä onnistuneesti. Viestejä mahtuu muistiin useita tuhansia, joten vaikka laite olisi pitkäänkin verkon kuulumattomissa, ei tieto silti pääse häviämään.

A1SW käyttää tiedonvälitykseen monia eri protokollia. Aplicomin itse kehittämistä laitteen tietojen välitykseen käytettävistä protokollista D-protokolla on kompaktein ja ylivoimaisesti käytetyin. D-protokollaa käytetään laitteen statuksen lähettämiseen palvelimelle. Statusviestin, eli niin sanotun snapshotin sisältö on käyttäjän konfiguroitavissa, ja se voi sisältää muun muassa paikkatiedon, nopeuden, GPS-ajan, kuljetun matkan, kiihtyvyyden, I/O:n tilat ja paljon muita laitteen itsensä tilaan liittyviä asioita. Protokollassa on lisäksi kenttä, jossa voidaan lähettää mitä vain tietoa, esimerkiksi joltain ulkoiselta laitteelta vastaanotettua dataa. D-protokolla on täysin binäärinen, eli harjaantumattomalle ihmissilmälle se näyttää vain heksanumeroilta.

Snapshottien lähettämiseen voidaan käyttää myös tekstimuotoisia TC- (Text Compact) ja TV- (Text Versatile) protokollia, joista TC on kompaktimpi ja TV helpommin ihmisen ymmärrettävä. Lisäksi on olemassa binäärimuotoiset F-protokolla FMS-CAN datalle, E-protokolla ajopiirturidatalle ja H-protokolla histogrammidatalle.

3.3.3 Asiakaskohtaiset versiot

A1-laitteen ohjelmistoista, niin GSM-moduulin kuin apuprosessorinkin osalta on olemassa myös asiakaskohtaisesti räätälöityjä versioita. Tällaiset erikoisversiot tulevat kyseeseen, kun asiakkaan tarve on niin yksilöllinen, ettei sitä saada toteutettua pelkästään perusohjelmiston konfigurointitiedostoa muuttamalla. Yleisimmät asiakaskohtaiset ohjelmistot käyttävät jotain erikoista protokollaa kommunikointiin palvelimen kanssa ja/tai kommunikoivat jonkin ulkoisen laitteen kanssa.

3.3.4 **Optiot**

A1-laitteen ohjelmisto sisältää jo vakiona laajat perusominaisuudet, mutta näiden ominaisuuksien lisäksi asiakas voi halutessaan ostaa käyttöönsä erilaisia lisäominaisuuksia ohjelmistoon eli ohjelmisto-optioita. Ohjelmisto-optio voi olla esimerkiksi ominaisuus, jolla A1 saadaan kommunikoimaan jonkin ulkoisen laitteen kanssa, kuten Garmin-navigaattorin, tai suorittamaan tietynlaista analysointia mitatuista arvoista, kuten histogrammeja. Optio voidaan aktivoida tuotannossa samalla kun asiakkaalle lähteviin laitteisiin ladataan koko muukin ohjelmisto tai jälkikäteen jo toimitettuihin laitteisiin.

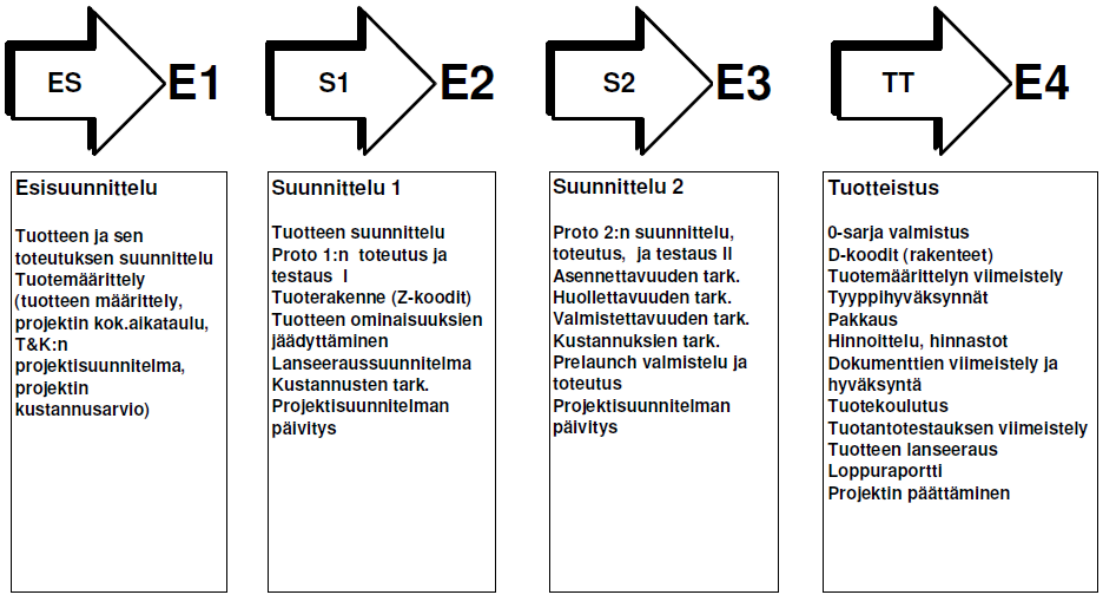
Ohjelmisto-optioiden lisäksi Aplicomin laitteissa on HW- eli ”rautaoptioita”. Ne ovat sellaisia laitteeseen erikseen ostettavia ominaisuuksia, jotka vaativat jonkin elektronisen komponentin tai toiminnallisuuden aktivoimisen laitteessa. HW-optioiden hallinnasta huolehtii apuprosessori, ja ne aktivoidaan samalla tavalla kuin ohjelmisto-optiot.

4 OHJELMISTOPROSESSI

4.1 Ohjelmistokehitys Aplicomilla

Aplicomin ohjelmistokehitys tapahtuu joustavalla ja projektikohtaisesti mukautuvalla tavalla. Perinteisistä ohjelmistokehityksen menetelmistä eniten käytetään inkrementaalista ja iteroivaa tapaa, mutta tärkein asia on tarkoituksenmukaisuus.

Tuoteprojekteissa, joissa ohjelmiston kehitys on sidoksissa laitteiston kehitykseen, käytetään lähellä vesiputousmallia olevaa tapaa ohjelmiston kehityksessä. Tämä on luonnollista, koska laitteiston kehitys ei voi olla samalla tavalla iteroivaa tai inkrementaalista, kuin ohjelmiston kehitys, vaan se on ensin suunniteltava kokonaan, sitten valmistettava kokonaan ja lopuksi testattava. Toki näitä kaikki vaiheet sisältäviä kierroksia voi olla ja käytännössä aina onkin muutama peräkkäin, ennen kuin tuote on valmis. Laitteiston kehityksen ennalta määrättyjen vaiheiden takia myös ohjelmiston kehitys tapahtuu pääosin laitteiston kehityksen aikataulun mukaisesti. Kokonaisuudessaan tuoteprosessi on määritelty Aplicomin toimintajärjestelmän kuvauksessa. Kuviossa 6 on esitetty tuoteprosessin toteutusprosessi.



Kuvio 6. Aplicomin tuoteprosessin toteutusprosessi (Toimintajärjestelmän kuvaus 2013.)

Projekteissa, joissa tehdään vain ohjelmistoon liittyvää kehitystä, voidaan käytettävät menetelmät valita vapaammin. Usein isommissa ja erityisesti asiakaskohtaisissa ohjelmistoprojekteissa käytetään inkrementaalista menetelmää, jossa uudet ominaisuudet toteutetaan yksi kerrallaan. Tällöin asiakas pääsee testaamaan tiettyä ominaisuutta jo ennen kuin kokonaisuus on valmis. (Haikala & Märijärvi 2000, 30.) Pienemmissä projekteissa, ja esimerkiksi yksittäisen ohjelmisto-option kehityksessä, tehokas menetelmä päästä nopeasti alkuun on iterointi, eli tehdään aluksi yksinkertainen runko ja kehitetään sitä eteenpäin.

4.2 Määrittely ja suunnittelu

Ensimmäisen korkean tason määrittelyn optiolle ja yleensäkin uusille ominaisuuksille ohjelmistossa tekee tuotehallinta. Se tekee myös päätöksen kehityksen aloittamisesta, jolloin vastuu siirtyy tuotekehitykselle. Yleensä tuotekehityksellä on hyvin vapaat kädet määritellä ominaisuus tarkemmin ja valita menetelmät sen toteuttamiseen. Toki todella paljon kommunikointia tuotekehityksen ja tuotehallinnan välillä on kehityksen kaikissa vaiheissa, ja myös tekninen määrittely tapahtuu suurelta osin yhteistyössä.

Jos tarvittava ominaisuus on selkeästi rajattu ja se käy sellaisenaan useammallekin asiakkaalle, on myös sen määritteleminen helpompaa. Jos taas kyseessä on monimutkaisempi asia tai perustelluista syistä joudutaan samalla ominaisuudella ratkaisemaan monen eri asiakkaan hieman erilaiset ongelmat, on ominaisuuden määrittely jo huomattavasti hankalampaa. Täytyy tehdä jonkinlainen kompromissi toteutukseen käytettävän ajan ja lopputuloksen geneerisyyden välillä.

Kun ominaisuus on saatu määriteltyä, voidaan alkaa suunnittelemaan toteutusta. Käytännössä määrittely ja suunnittelu tapahtuvat usein osittain limittäin, koska jotta tarvittava ominaisuus saadaan määriteltyä, voidaan joutua käyttämään aikaa myös sen tutkimiseen miten mikäkin mahdollisista vaihtoehdoista todellisuudessa toteutettaisiin. Ideaalitilanne olisi tietysti, että olisi niin hyvä määrittely valmiina toteutuksen suunnittelun alkaessa, ettei sitä tarvitsisi enää muuttaa. Tähän voidaan kohtuullisen yksinkertaisissa tapauksissa ja ennestään tuttujen asioiden kanssa toimittaessa päästäkin.

Asiakaskohtaisia ohjelmistoja tehtäessä määrittely on usein huomattavasti täsmällisempää ja selkeämmin erotettu kaikesta muusta kuin tavallisessa omien tuotteiden kehityksessä. Tämä johtuu tietysti siitä, että asiakas suoraan maksaa tuotteen kehityksen tehdyn tarjouksen perusteella ja kummallekaan osapuolelle ei saa tulla yllätyksiä kesken toteutuksen.

Perusvaatimuksina Aplicomin laitteen ohjelmiston uudelle osalle ovat – kuten yleensäkin kaikelle ohjelmistolle pitäisi olla – testattavuus, ylläpidettävyys ja valitun arkkitehtuurin mukaisuus. Sulautettu toimintaympäristö aiheuttaa kuitenkin myös sellaisia lisävaatimuksia, joita ei PC- tai web-ohjelmoinnissa tarvitse läheskään samalla tavalla huomioida. Näitä ovat kaikki suorituskykyyn liittyvät asiat, erityisesti muistin käyttö. Vaikka Aplicomin laitteiden suorituskyky päivittyykin säännöllisesti, ei muistia ikinä ole liikaa käytettävissä, ja on ajateltava, että joskus se loppuu kuitenkin. Koodi on siis kirjoitettava sekä testattavaksi että ylläpidettäväksi ja samaan aikaan vähän muistia kuluttavaksi. Luonnollisesti jos muistin käyttö on optimoitava viimeiseen asti, vähintään ylläpidettävyys kärsii huomattavasti. Koodista tulee tällöin helposti hyvin kryptistä luettavaa, koska jokainen if-lauseen ehto ja muuttujan nimikin vie muistia niin paljon, että sen vaikutus pitää huomioida.

Koska ohjelmisto tehdään Javalla, on hyvä ymmärtää myös vähintään perusasiat Javan toiminnasta sekä sen ominaisuuksista ja rajoituksista. A1:n GSM moduulissa on käytössä kevyt KVM-virtuaalikone ja Java Micro Edition (Java ME). Verrattuna PC-ympäristössä käytettyyn Java Standard Editioniin (Java SE), on ME suunniteltu käyttämään mahdollisimman vähän muistia ja muita sulautetuissa ympäristöissä rajoitetusti tarjolla olevia resursseja. Siinä on esimerkiksi vain osa SE:n tarjoamista luokkakirjastoista. Konkreettinen esimerkki tästä on Javan String-luokan split-metodin puuttuminen Java ME:stä.

4.3 Testaus

4.3.1 Yksikkötestit

Ohjelmiston testauksen matalin taso on yksikkötestaus. Yksikkötestaus tarkoittaa yksittäisen luokan tai moduulin testaamista ohjelmallisesti. Yleensä testitapaukset ovat funktio- tai rajapintakohtaisia. Periaate on antaa rajapinnalle tai funktiolle tietty syöte ja tutkia onko toiminta sen mukaista.

Yksikkötestit ovat parhaimmillaan silloin kun niitä käytetään yhtä aikaa ohjelmiston kehityksen kanssa. Ideaalitapauksessa ohjelmiston kehittäjä kirjoittaa aina uuden funktion tai rajapinnan tekemisen jälkeen sille mahdollisimman kattavan yksikkötestin. Tällä tavalla usein löydetään ensimmäiset yksinkertaiset virheet koodista, joiden paikantaminen myöhemmässä vaiheessa vaatisi huomattavasti enemmän työtä.

Hyvät yksikkötestit ovatkin tietynlainen vakuutus ohjelmistokehittäjälle siitä, että ohjelmisto todella toimii niin kuin on ajateltu. Valitettavan usein reaalimaailma ei aivan vastaa ideaalitilannetta, lähes aina on vähän kiire ja kunnolliset testit jäävät hyvin helposti kirjoittamatta. Jälkikäteen tehtynä niiden hyötykin on vähäisempi. Jos (ja kun) vikoja löytyy, niiden korjaaminen voi olla paljonkin työläämpää, kun funktioita ja rajapintoja käytetään jo jostain toisesta kohdasta koodia. Puhumattakaan siitä, että on monta kertaa palattava miettimään saman funktion toimintaa, kun sitä tehdessään olisi voinut lähes samalla vaivalla kirjoittaa myös testin.

Aplicomilla pidetään kiinni laadukkaasta yksikkötestaamisesta, sillä se on yksi ohjelmiston laadun tärkeimmistä kulmakivistä. Jos näin ei tehtäisi, maailmalle päätyisi aivan sataprosenttisen varmasti ohjelmistoja joissa on jokin vika.

4.3.2 Laboratoriotestit

Laboratoriotestit ovat laboratorio-olosuhteissa simuloituja käytännön tilanteita, joihin laite käytössä ollessaan joutuu. Laboratoriotesteissä voidaan tapauksesta riippuen testata jotain hyvinkin spesifistä ominaisuutta tai yksityiskohtaa laitteen toiminnassa, mutta

toisaalta voidaan järjestää myös erilaisia kuormitustestejä, joissa testataan kokonaisuuden toimintaa erilaisissa hallituissa olosuhteissa.

Laboratoriotesteillä usein pyritään toistamaan jokin kentällä havaittu ongelmatilanne ja siten rajaamaan ongelman aiheuttaja mahdollisimman tarkasti jatkotutkimuksia varten. Myös uutta ominaisuutta kehitettäessä ensimmäiset kokonaisuutta testaavat testit tapahtuvat usein laboratoriossa ennen kenttätesteihin siirtymistä.

4.3.3 Kenttätestit

Kenttätestit suoritetaan nimensä mukaisesti kentällä, eli laite kytketään mahdollisine lisälaitteineen esimerkiksi autoon ja sitä käytetään joko normaalisti tai tiettyä ominaisuutta kuormittaen. Yleensä kenttätestit ovat pitkäkestoisia ja niiden yksi tarkoitus onkin löytää pitempiaikaisessa käytössä mahdollisesti tapahtuvia virheitä. Toinen kenttätestien tärkeä tarkoitus on tutkia toimiiko laite tai jokin sen yksittäinen ominaisuus käytännössä niin kuin sen on sisällä pöydän ääressä ajateltu toimivan. Käytännön tilanteita voi olla usein hankala ennakoida, ja vaikka laitteen toiminnassa ei sinänsä vikaa olisikaan, voi jonkinlainen säätö olla tarpeen, jotta toiminta olisi halutunlaista.

4.3.4 Regressiotestit

Jokaiselle ohjelmistosta tehdylle julkaistavalle käännökselle ajetaan regressiotestit, eli testit joilla testataan ovatko uudet asiat ohjelmistossa rikkoneet jotain vanhoja asioita. Regressiotestit, eli niin sanotut ”smoke”-testit testaavat perustoiminnot ja sen, että juuri se yksittäinen jakeluun lähtevä tiedosto on toimiva, eikä käännöksessä ole tapahtunut mitään virheitä. ”Smoke”-testillä otetaan siis nimensä mukaisesti savut uudesta ohjelmistosta.

4.4 Tuotteistus

Tuotekehityksen osalta tuotteistuksen ehkä tärkein asia ohjelmisto-option tapauksessa on tekninen toteutus sille, että kyseistä ohjelmiston osaa voidaan käyttää vain kun ohjelmisto-optio on aktivoitu laitteelle. Tämä tehdään ohjelmallisesti A1 laitteen sisällä.

Lisäksi option toimintojen mahdollinen konfigurointi toteutetaan usein Aplicomin A1 SW Configurator-työkaluun. Se on graafinen työkalu, joka on tarkoitettu helpottamaan laitteen konfigurointiin liittyvien asetusten määrittämistä. Työkalu generoi XML-konfigurointitiedoston käyttäjän graafisella käyttöliittymällä säätämien asetusten mukaisesti. A1 SW Configurator-työkalulla voi myös helposti ladata uuden ohjelmiston ja konfiguraation A1-laitteeseen. Myös OTA-päivitys on helpointa tehdä laitteille A1 SW Configuratorilla.

Viimeinen tärkeä tuotekehityksen vastuulla oleva osa on asiakkaalle menevä tekninen dokumentointi. Optiosta ja sen käyttäjäkunnasta riippuen dokumentointi voi sisältää käyttöohjeita, tarkkoja toimintakuvauksia ja esimerkiksi protokollakuvauksia. Yleensä option toteuttanut ohjelmistosuunnittelija tekee siihen liittyvän teknisen dokumentaation, ja huolehtii että se tulee katselmoiduksi. Lopuksi dokumentointiosasto viimeistelee dokumentit julkaisua varten.

A1-laitteen ohjelmistosta julkaistaan vuosittain muutamia uusia versioita, jotka sisältävät päivityksiä, mahdollisia vikakorjauksia ja myös täysin uusia ominaisuuksia, kuten esimerkiksi Fuel Alert.

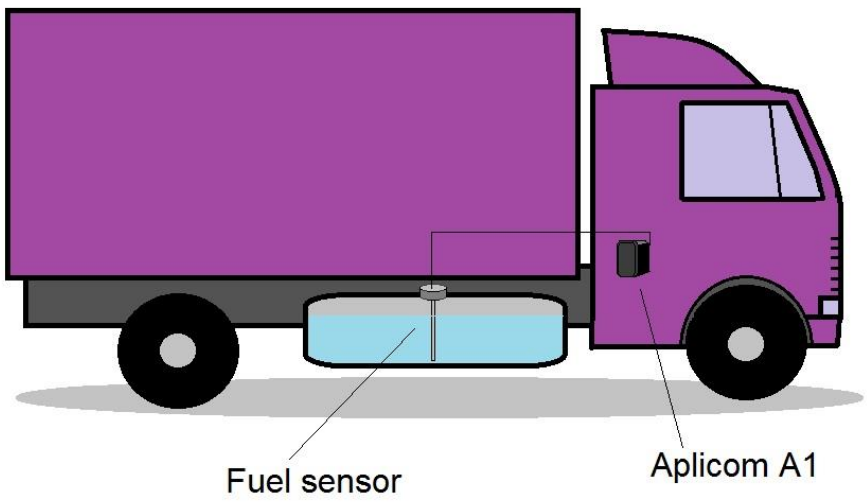
Uuden ohjelmistoversion julkaisu eli ”release” on tärkeä ja huolellisuutta vaativa osa ohjelmiston kehitysprosessia Aplicomilla. Asiakaskohtaiset julkaisut ovat aina yksilöllisiä, mutta esimerkiksi A1-laitteen ohjelmiston uusien versioiden julkaisut noudattavat samaa kaavaa. Ohjelmiston kehityksen ns. roadmappiin on kirjattu mitä asioita uudessa versiossa julkaistaan, ja tuotehallinta tilaa tuotekehitykseltä nämä uudet asiat sisältävän ohjelmistopaketin. Tuotekehityksen vastuulla on tuottaa paketti johon uudet ominaisuudet on kasattu, testata se ja tuottaa tarvittavat dokumentit, kuten release notet ja mahdollisesti päivittyneet käyttöohjeet ja muut asiakkaille ohjelmiston mukana toimitettavat dokumentit. Tuotehallinta suorittaa lopullisen julkaisun asiakkaille, kun kaikki hallinnolliset palaset itse teknisen tuotteen lisäksi ovat kunnossa.

5 FUEL ALERT

5.1 Lähtökohdat

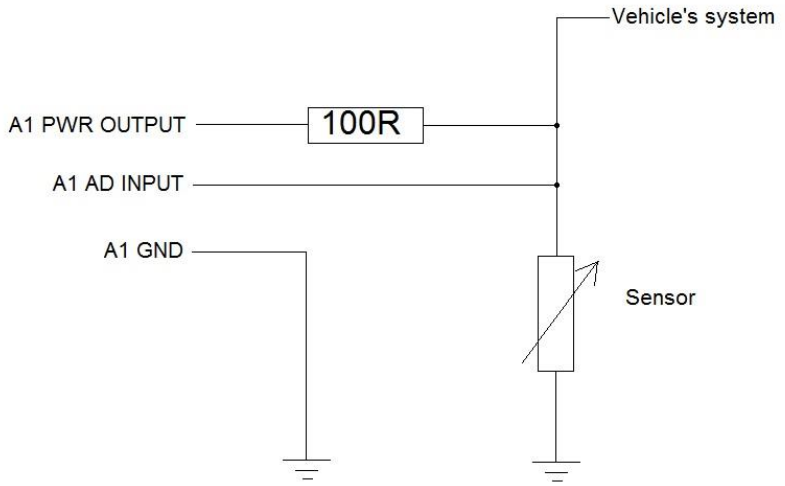
Polttoaineiden hinnat ovat viime vuosina nousseet ja nousevat edelleen muuallakin kuin Suomessa, ja monessa Euroopan maassa rikollisuus kasvaa ja sitä myötä myös polttoainevarkaudet ovat lisääntyneet. Asialla voi olla merkittävä taloudellinen vaikutus esimerkiksi kuljetusyritykselle, jolla on paljon kalustoa ympäri maata, tai työkoneurakoitsijalle, jonka kaivinkoneet ja puskutraktorit usein seisovat työmailla yöt ilman vartiointia ja siten ovat houkutteleva kohde polttoainevarkaille. Polttoaine on yleensä muuttuvista kustannuksista suurin kuluerä kuljetusyrityksille ja työkoneiden omistajille, joten asennetun laitteen kustannukset voidaan säästää moninkertaisesti, kun varkaudet saadaan vähenemään.

Ajatus oli kehittää ohjelmisto-optio, jonka avulla on mahdollista havaita jos ajoneuvon tankista häviää polttoainetta sen ollessa paikallaan. Pääasiallinen käyttötarkoitus olisi siis polttoainevarkauksien havaitseminen ja sitä kautta niiden estäminen. Kuviossa 7 on kuvattu A1-laite tarkkailemassa kuorma-auton polttoainetankkia.



Kuvio 7. Fuel Alert (Fuel Alert SW Option 2014.)

Fuel Alertin kanssa oli tarkoitus käyttää ajoneuvon omaa polttoaineanturia, joten mitään erillisiä antureita ei tarvittaisi. A1-laite kytkettäisiin polttoaineanturiin rinnan ajoneuvon oman järjestelmän kanssa. Kuviossa 8 on periaatteellinen kytkentäkaavio anturin kytkennästä.



Kuvio 8. Polttoaineanturin kytkentä (Fuel Alert SW Option 2014.)

Tilaus Fuel Alert-option toteuttamisesta tuli normaalisti tuotehallinnalta tuotekehitykselle, ja työn katsottiin olevan sopivasti muusta kehityksestä erillään oleva pala, jota juuri työt aloittanut harjoittelija voisi rauhassa tehdä eteenpäin sitä mukaa kun oppii.

5.2 Määrittely

Fuel Alertin määrittely alkoi siltä pohjalta, että oli tarve saada selville milloin polttoainetta häviää, ja nimenomaan niin että voidaan tulkita sen olevan varkaus.

Tiedossa oli, että ajoneuvon omalta anturilta saatava arvo heittelee huomattavasti toimintaperiaatteestaan johtuen ajoneuvon liikkuessa, joten liikkeen aikana on turha yrittääkään tulkita mikä heilahduksista olisi varkaus. Ja onhan ajoneuvon liikkuessa polttoaineen varastaminenkin melko epätodennäköistä.

Alkuvaiheessa oli selvillä myös se, että tarkoitus on tukea kahden mallisia polttoaineantureita jotka molemmat ovat resistanssin muutokseen perustuvia. Ero antureiden välillä

tulee siitä, kumpaan suuntaan anturin resistanssi muuttuu polttoaineen pinnan tason muuttuessa. Voi siis olla antureita joiden resistanssi nousee kun polttoaineen pinta nousee, tai päinvastoin.

Resistanssin muutokseen perustuvan anturin arvo saadaan selville mittaamalla sen yli jäävää jännitettä. Anturi kytketään A1-laitteen AD-sisääntuloon ja jännitettä mittaa apuprosessori. Se välittää tiedon GSM-moduulille ja siten tieto anturin arvosta saadaan Java-ohjelmiston käyttöön. Jännitteenmittaustoiminnallisuus on A1-laitteessa jo olemassa oleva perusominaisuus ja sen toteuttaminen ei sisältynyt tähän työhön.

5.3 Suunnittelu

Fuel Alertin suunnittelussa käytettiin perinteisistä ohjelmistosuunnittelun menetelmistä eniten inkrementaalista ja iteratiivista lähestymistapaa. Aluksi siis tehtiin hyvin yksinkertainen versio, jota sitten kehitettiin eteenpäin lisäämällä uusia asioita ja parantamalla jo olemassa olevia.

Ennen yhdenkään Fuel Alertiin liittyvän koodirivin kirjoittamista ja itse toteutuksen suunnittelua oli aiheellista tutustua A1:n AD-mittauksen toimintaan, ja siihen miten raaka AD-muuntimelta tuleva arvo on suodatettu ennen kuin se on Javan käytettävissä ja mitä käsittelyjä sille tehdään Javan puolella. Myös olemassa olevia toimintoja jännitteen mittaamiseen liittyen tutkittiin, kuten tapahtumien generoitumiseen jännitetasojen vaihdellessa AD-sisääntulossa.

Suunnittelun alussa, kun ei vielä juurikaan ollut kokemusta ja ymmärrystä miten mikäkin asia kannattaisi toteuttaa, oli helpottavaa ajatella, että periaatteessa koko toiminnan ydin on hyvin yksinkertainen vertailu kahden luvun välillä; onko polttoainetta enemmän tai vähemmän kuin pitäisi.

Työn edetessä haastekin lisääntyi. Vaikka itse vertailu onkin yksinkertainen, niin se mitä sen tuloksena saadulla tiedolla voi ja kannattaa tehdä, on vielä asia erikseen. Lisäksi kaikkien Fuel Alertin toimintaan vaikuttavien asioiden ymmärtäminen ja huomioiminen

monimutkaisti suunnittelua, eikä kaikkia asioita voinut kokonaisuudessaan tietääkään heti aluksi.

Koska ympäristö jossa ohjelmisto toimii ja myös Fuel Alertin koodi hoitaa omaa osaansa on ajoneuvossa oleva sulautettu järjestelmä, on erilaisia huomioon otettavia tilanteita todella paljon enemmän kuin esimerkiksi PC- tai web-ympäristöissä. Vaikka Fuel Alertin kanssa käytettävä anturointi on yksinkertainen ja kohtuullisen luotettava, vaikuttaa jo resistanssin yli mitattavaan jännitteeseen moni asia jota ei voi jättää huomiotta. Esimerkiksi sellaiset käytännön asiat kuten akun tyhjeneminen ja täyttyminen ja auton virtojen päälle laittaminen vaikuttavat mitattuun tulokseen. Myös se milloin ajoneuvo liikkuu, on Fuel Alertin tapauksessa tärkeä tieto, koska loiskuvasta polttoaineenpinnasta on hankala saada mitattua järkeviä arvoja. Toki monenlaista keskiarvoistusta voidaan tehdä, mutta koska tankin tilavuus saattaa olla useita satoja litroja ja tarkoitus on havaita mahdollisimman pieni polttoaineen pinnan muutos, on mittauksen oltava tarkka myös lyhyellä ajalla. Tarkoituksenmukaista on siis keskittyä mittaamaan paikallaan olevan ajoneuvon polttoaineen määrää.

Kun oli saatu pohdittua miten jokin asia kannattaa tehdä ja mitä siihen liittyen pitää huomioida, ei se vielä riittänyt kovin pitkälle. Koska ymmärrys Fuel Alertiin välittömästi liittyvä koodin ympärillä olevan koodin toiminnasta oli hyvin rajallista, tekemistä oli paljon myös sen selvittämisessä, miten käytännössä ajatukset kirjoitetaan koodiksi. Sen jälkeen vasta oli mahdollista kokeilla, valitettavasti usein käytännössä vain laboratorio-olosuhteissa, oliko ajatuksissa järkeä.

5.4 Toteutus

Kun se oli selvillä, mitä toteutettavan ohjelmiston osan tulisi kyetä tekemään ja jonkinlainen arvaus myös siitä miten se tehdään, oli aika aloittaa toteutus. Alku oli hyvin kankeaa, kaikki asiat olivat uutta kehitysympäristöstä lähtien, puhumattakaan itse ohjelmiston rakenteesta ja toiminnasta. Ensimmäinen viikko menikin Eclipse-kehitysympäristön asetuksia säätäessä ja lisäosia asennellessa, samalla vähitellen perehtyen myös ohjelmistoon. Täyttä hepreaa se ei ollut, olihan takana yhden opintojakson verran Java-

ohjelmointia koulussa. Ilman sitä olisi tullutkin todella rankka aloitus työskentelylle. Toki suuri osa koodista oli sellaista, jonka tarkoitus ei heti avautunut, mutta yllättävää oli kuinka paljon siitä kuitenkin ymmärsi ja pystyi arvaamaan ja päättelemään. Eivät koulussa opetetut jutut olleetkaan ihan eri planeetalta kuin työelämän Java.

Suurin haaste ohjelmointityötä aloittaessa ei ollutkaan Fuel Alertin logiikan toteutus, vaan se miten se liitetään muuhun, jo olemassa olevaan koodiin. Kysymyksiä heräsi paljon, ja niiden vastaukset herättivät vielä lisää kysymyksiä. Mistä löytää AD-muuntimien arvot, miten ihmeessä generoidaan eventtejä ja miten tämä voisi olla jotenkin konfiguroitavissa flash-muistin nurkalla olevalla xml-tiedostolla?

Onneksi kaikkiin asioihin ei tarvinnutkaan heti löytyä vastausta, ja yksinkertaisemmat asiat selvisivät kohtuullisen nopeasti kun vain pyysi asiantuntija-apua. Toki se, että osaa mekaanisesti kirjoittaa tai kopioida jostain pätkän koodia on vielä kaukana siitä, että oikeasti ymmärtäisi mitä tapahtuu, mutta jostain on aloitettava.

Ensimmäiset askeleet itse ohjelmointityössä liittyivät nimenomaan AD-muuntimien jännitteiden selvittämiseen ja siihen miten niistä saisi tulkittua parhaiten polttoaineen pinnankorkeuden vaihtelut. AD-muuntimen arvo on onneksi jo hyvin suodatettua kaikenlaisilta häiriöiltä siinä vaiheessa, kun se on Javalla käytettävissä. Se tosin ei ole vielä voltteja eikä millivoltteja, eli jotta se saadaan muunnettua jännitteeksi, se pitää kertoa sopivalla luvulla, joka riippuu mittausalueesta (5, 10, 20 tai 40 volttia).

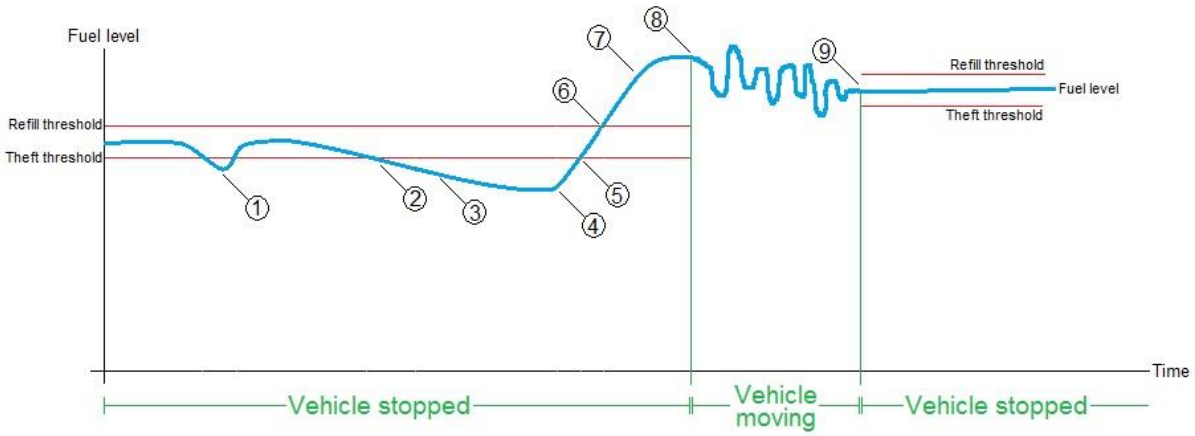
Seuraava merkittävä tekijä Fuel Alertin kannalta oli liikkeen tunnistaminen. Periaate tunnistamisessa on, että apuprosessori tarkkailee kiihtyvyysanturia ja ilmoittaa Java-prosessorille liikkeen alkaneeksi tai pysähtyneeksi. A1SW:n sisällä olevan tapahtuma-toiminto-logiikka generoi ilmoituksen perusteella tapahtuman, jonka avulla tieto liikkeellelähdöstä ja pysähtymisestä oli helposti myös Fuel Alertiin liittyvän koodin saatavilla.

Tämän tiedon perusteella piti asettaa pinnan tarkkailu päälle tai pois, ja siirryttäessä tilasta toiseen piti myös tallentaa anturin arvoja flash-muistiin tai lukea sieltä vanha arvo. Alussa tämäkin osa toimintaa oli luonnollisesti hyvin yksinkertainen, ja laajeni ja kehittyi vähitellen.

Yksi tärkeä huomioitava asia oli lisäksi jännitteensyötön muutokset. Kun ajoneuvon sytytysvirta kytketään päälle, sen oma järjestelmä syöttää polttoaineanturille käyttöjännitteen, ja kun se kytketään pois, pitää A1-laitteen huolehtia anturin sähköistämisestä. Tämä yksinkertainen vaihtologiikka voidaan tehdä myös A1:n konfiguraatiolla, joten sitä ei lähdetty erikseen järjestämään Fuel Alertin koodiin. Koodissa piti kuitenkin huomioida, että kun jännitteensyötön vaihto tapahtuu, erittäin todennäköisesti anturin näyttämä jännitetaso muuttuu niin paljon, että se vaikuttaa merkittävältä muutokselta polttoaineenpinnassa. Samantyylinen logiikka kuin liikkeellelähdön ja pysähtymisen kanssa piti siis tehdä myös sytytysvirran päälle- ja poiskytkemisen varalle.

Jotta kaikki mahdolliset skenaariot ajoneuvon liikkumisen ja erilaisten sähkönsyöttötilanteiden kanssa olisi katettu myös Fuel Alertin puolesta, piti kehittää paljon käyttäjältä piilossa olevaa toimintaa. Käyttäjän säädettäväksi tehtiin kuitenkin paljon parametreja, jotta Fuel Alertin toiminta voidaan optimoida tarvittaessa vaikka jokaiselle ajoneuvolle erikseen.

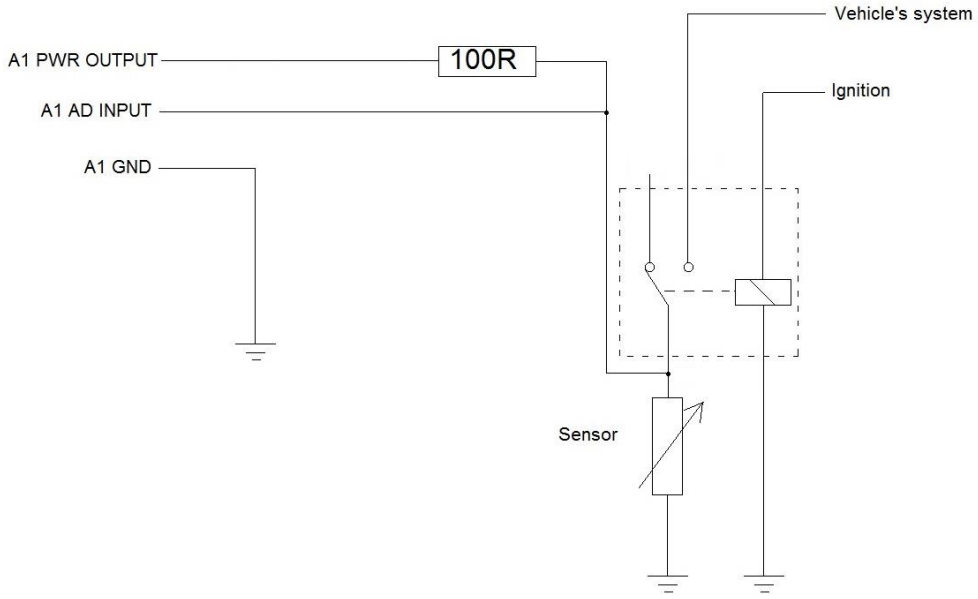
Fuel Alertin lopullinen toimintalogiikka on perusperiaatteeltaan seuraavanlainen: se seuraa ajoneuvon liikkeitä kiihtyvyysanturilta tulevien tietojen perusteella, ja kun se havaitsee ajoneuvon pysähtyneen, se odottaa hetken polttoaineen pinnan loiskunnan loppumista ja tallentaa sen hetkisen polttoaineanturin arvon Flash-muistiin. Sen jälkeen se aloittaa anturin arvon seurannan konfiguraation määrittelemällä tavalla. Mikäli määritetyt ehdot toteutuvat, eli polttoaineen pinta joko laskee tai nousee riittävän paljon ja pysyy rajan ylittävässä arvossa riittävän kauan, Fuel Alert generoi hälytystapahtuman A1SW:n tapahtumakoneistolle. Koneisto toimii konfiguraation mukaisesti, ja suorittaa määritellyn toiminnon, eli esimerkiksi lähettää snapshotin palvelimelle tai tekstiviestin auton omistajalle. Graafinen esitys toiminnasta on kuviossa 9.



Kuvio 9. Fuel Alertin toiminta (Fuel Alert SW Option 2014.)

1. Polttoaineen pinta laskee rajan alapuolelle, mutta nousee takaisin sen yläpuolelle ennen kuin laskuri saavuttaa raja-arvonsa.
2. Polttoaineen pinta laskee uudestaan rajan alapuolelle.
3. Laskuri saavuttaa raja-arvonsa ja varkaus-tapahtuma generoituu.
4. Tankkaus alkaa.
5. Pinta nousee varkaus-tason yläpuolelle
6. Pinta saavuttaa tankkaus-tason.
7. Laskuri saavuttaa raja-arvonsa ja tankkaus-tapahtuma generoituu.
8. Ajoneuvo alkaa liikkua ja polttoaineen pinnan tarkkailu keskeytetään.
9. Liike loppuu, uusi vertailuarvo tallennetaan ja polttoaineen pinnan tarkkailu jatkuu.

Mikäli A1-laitteen kytkeminen aiheuttaa virhettä ajoneuvon omaan polttoainemittariin, käytetään tarkoitukseen sopivaa relettä, jolla kytketään A1-laite irti ajoneuvon järjestelmästä sytytysvirran ollessa kytketty päälle. Kuviossa 10 on esitetty kytkentä lisäreleen kanssa.



Kuvio 10. Polttoaineanturin kytkentä releen kanssa (Fuel Alert SW Option 2014.)

5.4.1 Työkalut ja menetelmät

Eclipse-ympäristö

Ohjelmiston kehitysympäristönä käytössä oli erityisesti Java-ohjelmoijien keskuudessa yleisesti suosittu ja myös kaikilla Aplicomin Java-kehittäjillä käytössä oleva Eclipse IDE (Integrated Development Environment). Siihen on suuren ja aktiivisen käyttäjäkunnan ansiosta saatavilla paljon erittäin hyviä ja laajasti käytössä olevia ns. open-source-lisäosia moneen tarkoitukseen.

Ohjelmiston kääntäminen

Ohjelmiston kääntäminen tapahtuu Apache Ant-käännöstyökalulla, joka on open-source-projektina tehty Javaan perustuva työkalu. Eclipseen on saatavilla lisäosana Ant UI joka on Ant-käännösskriptitiedostojen editointiin sopiva käyttöliittymä. Ant-käännösskriptien nimet alkavat aina sanalla ”build” ja ne ovat XML-tiedostoja.
Skriptitiedosto määrittää mm. sen mitä luokkia ja ulkoisia kirjastoja otetaan mukaan käännökseen. Ohjelmakoodin seassa voi olla esikääntäjäkomentojen (ifdef, ifndef jne.)

avulla käyttöön otettavia tai käytöstä poistettavia osia, ja ne on myös määritelty skriptitiedostossa. Kun skriptiä sitten aletaan ajaa, valitaan vain rasti ruutuun -periaatteella mitkä osat otetaan mukaan. Samalla periaatteella voidaan muitakin asetuksia valita päälle tai pois, kuten ajetaanko yksikkötestit, poistetaanko vanhat käännökset kansioista ja niin edelleen. Ant-työkalussa näitä rasti ruutuun -asetuksia kutsutaan targeteiksi.

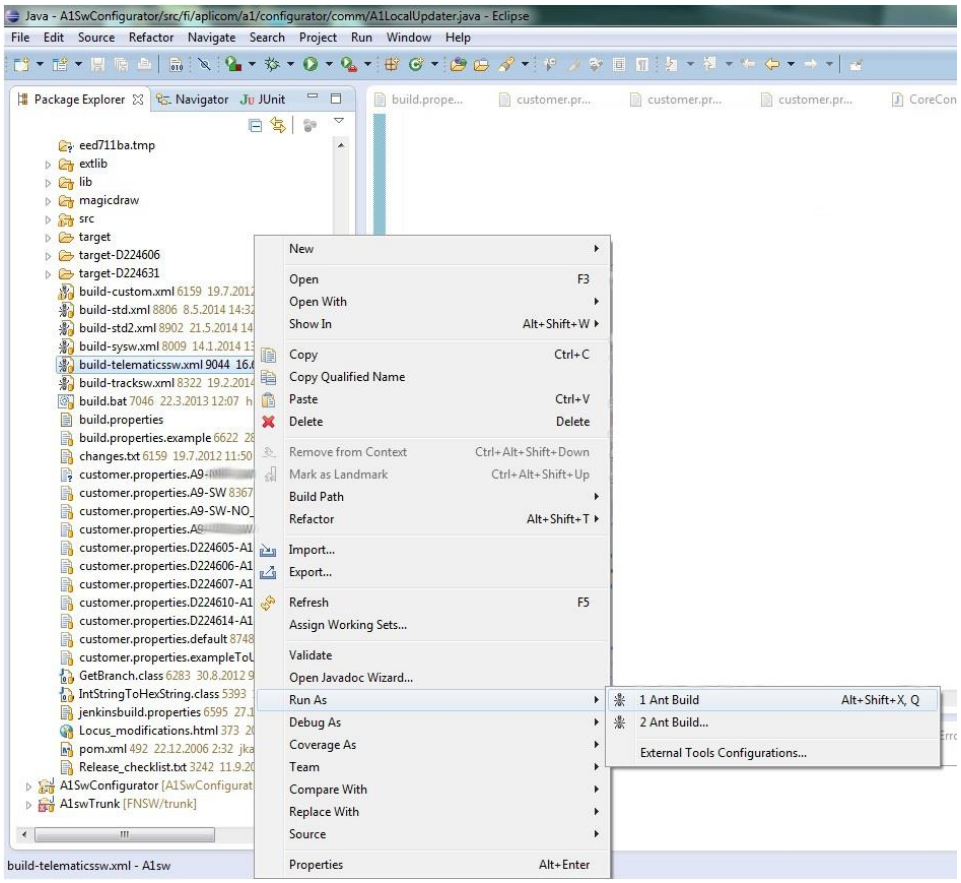
Itse kääntämiseen liittyvien asetusten lisäksi skriptissä määritellään myös se, miten tuloksena syntyvät tiedostot paketoidaan ja nimetään. Skripteillä voidaan määritellä hyvinkin monimutkaisia toimintoja ja ehtoja käännökseen, ja esimerkiksi A1 MAX SW:n kääntämiseen käytetyssä skriptitiedostossa on yli 1300 riviä.

Obfuskointi

Yleinen tapa estää niin sanottua reverse-engineeringiä, eli lähdekoodia tai binääritiedostoja tutkimalla ohjelmiston toiminnan selvittämistä, on tiedostojen obfuskointi, eli tarkoituksella niiden saattaminen mahdollisimman vaikeaksi tai jopa mahdottomaksi ihmiselle ymmärtää.

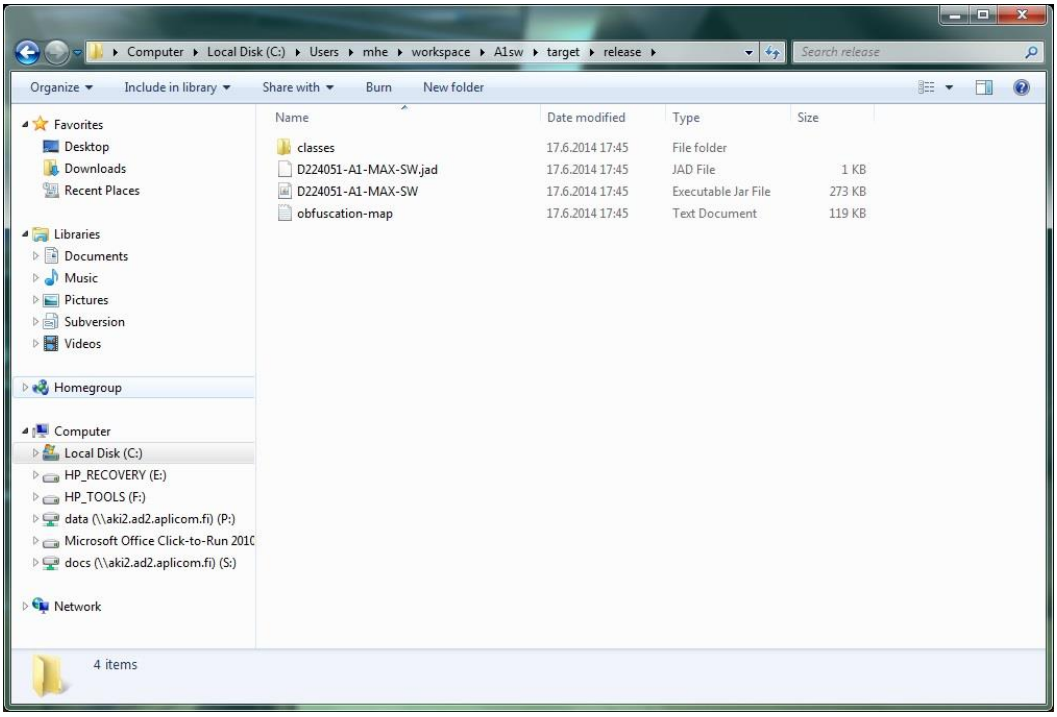
Myös Aplicomin A-sarjan ohjelmistot obfuskoidaan oletusarvoisesti jokaisessa käännöksessä. Obfuskoinnin suorittaa Eclipseen integroitava työkalu, joka on kehitetty erityisesti ohjelmakoodin optimointiin ja obfuskointiin. Saman työkalun avulla saadaan myös esimerkiksi jätettyä käyttämättömät luokat pois lopullisesta paketista, joka luonnollisesti säästää erityisesti sulautetussa ympäristössä kallisarvoista muistia. Asetukset työkalulle asetetaan samassa Ant-skriptitiedostossa kuin muutkin käännöksen asetukset. Siellä voi myös manuaalisesti estää haluttujen luokkien obfuskoinnin tai pakottaa niiden mukana pitäminen vaikka kääntäjän mielestä niitä ei ikinä käytettäisi. Tätä ominaisuutta hyödynnetään muun muassa silloin, kun on tarve käyttää ohjelman ajon aikana dynaamisesti käyttöön ladattavia luokkia.

Ohjelman kääntäminen aloitetaan klikkaamalla Eclipsen projektinäkymässä näkyvää build-tiedostoa hiiren oikealla painikkeella ja valitsemalla Run As (ks. kuvio 11). Se tuleeko käännettävästä ohjelmasta MAX vai TRAX tuotteen ohjelmisto, valitaan sillä mikä skriptitiedosto ajetaan.



Kuvio 11. Käännöksen aloitus Eclipsessä

Koko käännösprosessi kestää kaiken kaikkiaan reilut kymmenen minuuttia, sillä osassa yksikkötesteistä on hyvinkin pitkiä viiveitä, joita tarvitaan kun testataan esimerkiksi tiettyjä ajastuksia. Varsinainen ohjelman käännös kestää hieman asetuksista riippuen muutamasta kymmenestä sekunnista minuuttiin. Käännöksen jälkeen kaikkein eniten kiinnostavat tiedostot, eli laitteeseen ladattavat jar- ja jad-tiedostot löytyvät Eclipsen workspace-kansiosta käännösskriptissä määritellystä kansiosta, joka tässä tapauksessa on A1sw/target/release (ks. kuvio 12). Samassa kansiossa on myös obfuskoinnin tulkkaamiseen tarkoitettu obfuscation-map-tekstitiedosto, ja omassa kansiossaan käännökseen mukaan otetut ja obfuskoimattomat luokat.



Kuvio 12. Käännöksen tuloksena syntyneet jad- ja jar-tiedostot

Ohjelmiston lataaminen laitteeseen

Ohjelmisto voidaan ladata laitteeseen joko käyttämällä manuaalisesti komentorivillä tai Windowsin resurssienhallinnalla GSM-moduulin omaa tiedostonsiirto-ohjelmisto MES:iä, tai A1 SW Configuratorilla. Aplicomilla on totuttu käyttämään jälkimmäistä, koska se on monen mielestä helpompi ja samalla työkalulla voi myös muokata kätevästi konfiguraatioita. Konfiguraation muokkaukseen ei keskitytä tässä, vaan ainoastaan ohjelmiston ja konfiguraation lataamiseen laitteeseen.

Laitteen tulee olla kytketty tietokoneeseen COM-porttiin tarkoitukseen olevalla kaapelilla. Kaapelin toinen pää kytketään A1-laitteen COM1-porttiin. Lisäksi laitteeseen tulee olla kytketty vähintään käyttösähköt, jotta ohjelmiston lataus voidaan tehdä. Liitteessä 1 on tarkemmat ohjeet ohjelmiston lataamiseen.

Debuggaus

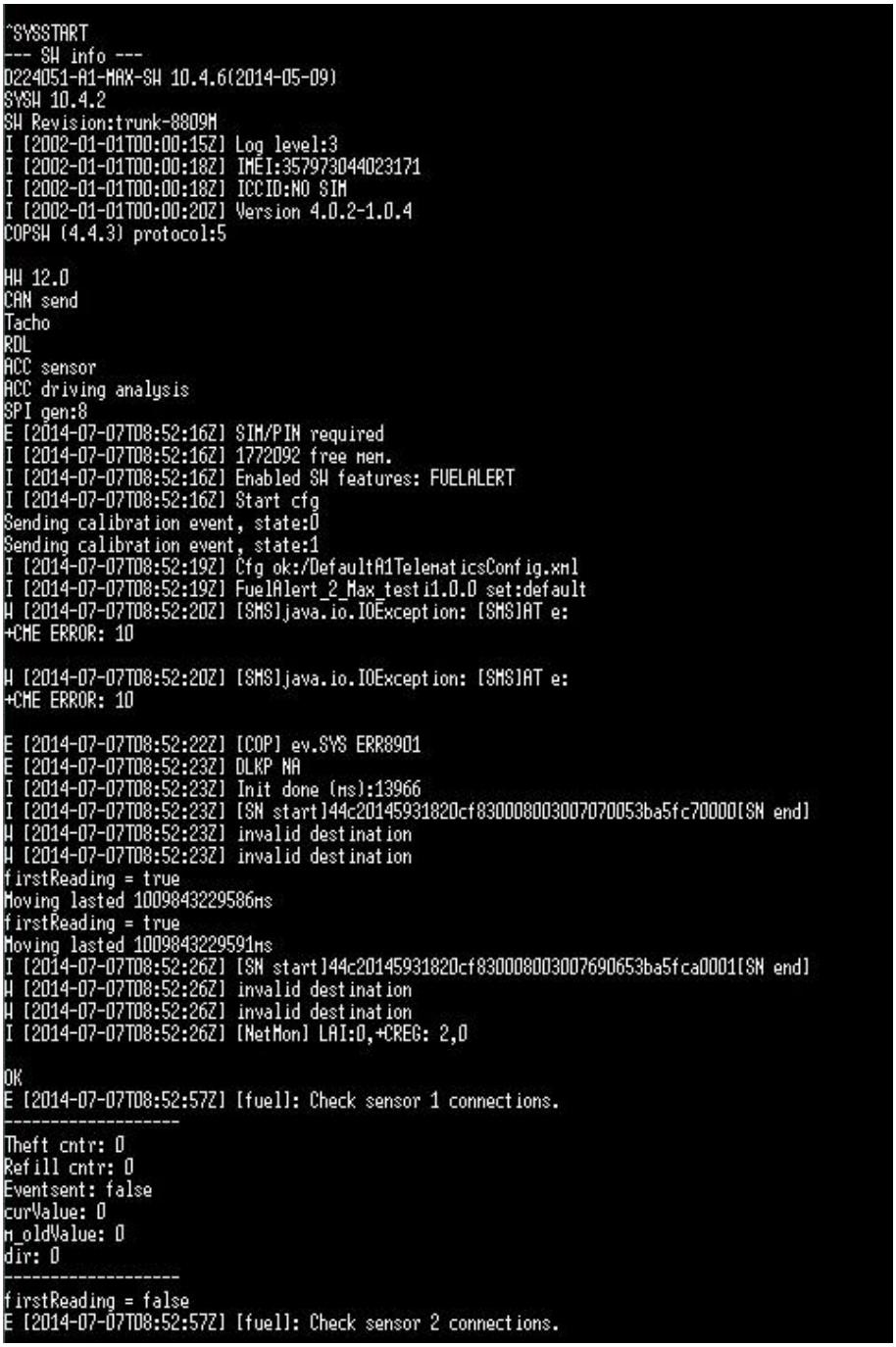
Koska harvoin ohjelmat toimivat kerralla oikein, työtä helpottaa huomattavasti jos käytettävissä on jokin tapa debugata, eli tutkia ohjelman toimintaa tai toimimattomuutta sen ollessa ajossa.

A1-laitteen tapauksessa käytettävissä ei ole rajapintaa, jolla voisi suoraan nähdä muuttujien tilat tai askeltaa ohjelman suoritusta, eikä myöskään omaa näyttöä, johon voisi tulostaa tietoja ohjelma suorituksesta.

Onneksi laitteessa on kuitenkin monta sarjaporttia, joista yksi on voitu varata debug-lokituskäyttöön. Kun A1:n COM2-porttiin kytkee tarkoitukseen sopivan kaapelin ja yhdistää sen tietokoneen sarjaporttiin, voidaan terminaaliohjelmalla (esimerkiksi putty tai Teraterm) lukea laitteen lähettämää dataa. Ohjelmakoodissa voi sitten lokitusta hallinnoivan olion kautta kirjoittaa lokiin mitä ikinä tarvitseekin, ja sopiviin paikkoihin koodia lokituksia lisäämällä voi saada kohtuullisen hyvän kuvan ohjelman toiminnasta ja mahdollisista ongelmista. Toki tämä edellyttää, että laite toimii sen verran, että COM2-portti ja lokitus ovat käytettävissä.

Kuvion 14 lokista näkyy laitteen käynnistys. Aivan alussa on lueteltu ohjelmistojen versiot ja muun muassa laitteen yksilöivä IMEI-koodi. Noin puoleen väliin kuvaa lokissa on laitteen initialisoinitiin liittyviä lokituksia. Niistä voi päätellä esimerkiksi sen että SIM-kortti ei ole paikallaan laitteessa. Ensimmäinen Fuel Alertiin suoranaisesti liittyvä teksti on ”firstReading = true”. Se kertoo erään boolean-tyyppisen muuttujan tilan, joka oli kuvan tilanteessa tärkeä tieto siitä toimiiko kaikki oikein.

A1-laite tallentaa debug-lokia myös omaan Flash-muistiinsa, ja se voidaan ladata kentällä olevasta laitteesta ilmateitse tutkittavaksi. Tämä voi olla ratkaiseva apu kaukana maailmalla ilmeneviä ongelmia selvitettäessä.



Kuvio 13. A1-laitteen debug-lokia Fuel Alertin ollessa käytössä

5.4.2 **A1 SW Configurator**

Yksi vaatimuksista oli, että Fuel Alertin parametrit (anturin tyyppi, AD-kanava, hälytykseen vaadittava jänniteraja, tallennuksen viive liikkeen loputtua ja hälytykseen vaadittavien rajan ylitysten määrä) piti voida asettaa Aplicomin A-sarjan laitteiden konfigurointiin tarkoitetulla A1 SW Configurator –ohjelmalla. Ensimmäistä versiota varten tehdyt

lisäykset Configuratoriin eivät sisältyneet tähän työhön, mutta toisen version vaatimat muutokset, eli parametrien säädöt toisella tankille ja pieni hienosäätö ulkoasuun sisältyivät.

A1 SW Configurator on Java SE:llä tehty ohjelma, jolla voidaan luoda ja muokata konfiguraatioita kolmella eri tavalla, joista käyttäjälle yksinkertaisin on wizard-vaihtoehto. Siitä on karsittu monimutkaisimmat asiat pois kokonaan eikä sillä voi esimerkiksi asettaa Fuel Alertin parametrejä. Kaikkia asetuksia voi säätää Advanced-tilassa, ja jos raaka XML-data tuntuu luontevimmalta, myös sitä pääsee suoraan muokkaamaan A1 SW Configuratorilla. Kaikki eri tavat luoda konfiguraatioita tuottavat saman lopputuloksen, eli XML-muotoisen konfigurointitiedoston, jossa kaikki tarvittavat asetukset on määritetty.

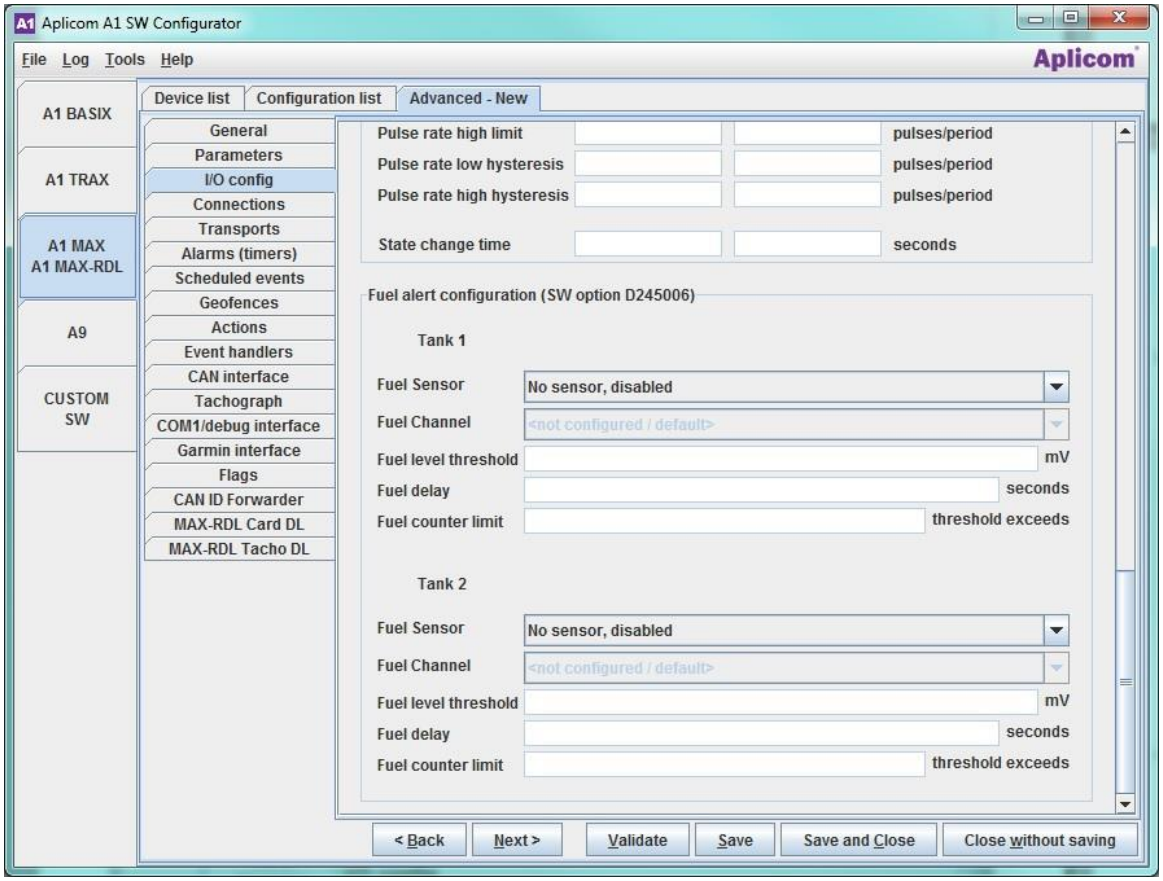
Konfiguraatioiden muokkaamisen lisäksi tärkeä ominaisuus on ohjelmiston ja konfiguraation lataaminen laitteeseen. Configuratorilla voi ladata ohjelmiston ja konfiguraatiotiedoston (tai useamman vaihtoehtoisen tiedoston) laitteeseen sekä paikallisesti että etänä OTAP-toiminnon avulla. Configuratorilla voi myös poistaa kaiken datan A-sarjan laitteista, ja se onnistuu myös sekä paikallisesti että etänä.

Fuel Alert-option konfigurointiin on yhteensä kymmenen parametria, viisi molemmille tankeille. Parametrit näkyvät kuviossa 14. Ensimmäinen Configuratorin listassa olevista parametreista on anturityyppi, jonka valinta vaikuttaa siihen onko kyseisen tankin valvonta päällä vai ei. Jos alasvetovalikosta on valittu ”No sensor”, eli ei anturia, ei myöskään muita parametreja kyseisen tankin osalta pääse säätämään eikä tankin valvonta ole päällä.

Listalla seuraava, eli ”Fuel Channel” tarkoittaa A1-laitteen AD-kanavaa, johon polttoaineanturi kytketään. ”Fuel level threshold” on raja jota enempää polttoaineanturin arvo ei saa muuttua tai seuraa hälytys. Arvo annetaan millivoltteina.

”Fuel Delay”-parametrillä säädetään aikaa, joka odotetaan ajoneuvon pysähtymisen jälkeen ennen kuin polttoaineenpinnan tarkkailua jatketaan. Jonkin verran viivettä täytyy luotettavan toiminnan varmistamiseksi olla, sillä isossa tankissa polttoaine voi aaltoilla pitkään pysähtymisen jälkeen.

”Fuel counter limit”- parametrillä voidaan säätää sitä, montako kertaa ero polttoaineanturin mitatun arvo ja vertailuarvon välillä saa ylittää ”Fuel level threshold”-parametrillä asetetun arvon ennen kuin tehdään hälytys.



Kuvio 14. A1-laitteen konfigurointiin käytettävä A1 SW Configurator-ohjelma. Alareunassa on osuus Fuel Alertin parametrien konfigurointiin

5.4.3 **Katselmoinnit**

Ennen Fuel Alertin ensimmäistä julkaisua järjestettiin dokumentti- ja koodikatselmoinnit Aplicomin käytänteiden mukaisesti. Ensimmäisessä katselmoitiin option kuvaava ”application note”-tyyppinen dokumentti. Mitään vakavaa ei katselmoinnin ensimmäisellä kierroksella dokumentista löytynyt, mutta sen verran kuitenkin katselmointiin osallistuneiden mielestä muutoksia kaipaavia pienempiä asioita, että uusintakatselmointi päätettiin järjestää. Toisella kerralla löytyi enää yksittäisiä sanamuotoihin liittyviä asioita, joita päätettiin muuttaa, joten dokumentti hyväksyttiin sovituin muutoksin julkaistavaksi.

Koodikatselmoinnissa osallistujia oli selvästi vähemmän kuin dokumenttia katselmoitaessa, itse tekijän lisäksi vain Aplicomin johtava ohjelmistosuunnittelija. Katselmointitilaisuudessa katsottiin koodi läpi, ja etukäteen kommentteja herättäneet asiat tutkittiin tarkemmin. Siihen nähden minkälaisella kokemuksella koodi oli kirjoitettu, korjattavaa löytyi yllättävän vähän. Mitään kuolettavan vakavaa ei löytynyt, mutta toki parannettavaa kuitenkin oli. Moni korjaamista vaativa asia oli sellainen että jonkin asian toteuttamiseen oli järkevämpikin tapa kuin mitä oli käytetty. Osa katselmoinnissa esille tulleista asioista päätettiin laittaa harkintaan seuraavaa versiota kehitettäessä, ja osa niistä aikanaan toteutuikin.

5.5 Testaus

5.5.1 Testauksen jakautuminen

Fuel Alertin testauksesta tehtiin testaussuunnitelma, johon kirjattiin vaatimukset tehtäville testeillä. Testaus jakautui kehitystyön aikana pääasiassa kolmeen osaan, yksikkötesteihin, laboratoriotesteihin ja kenttätesteihin.

5.5.2 Yksikkötestaus

Yksikkötestien tekeminen on luonnollisinta ja ennen kaikkea hyödyllisintä nimenomaan yhtä aikaa ohjelmiston kehityksen kanssa, ja ideaalitilanteessa aina yhden funktion tai moduulin kirjoituksen jälkeen sille tehdään yksikkötesti jolla tutkitaan onko toiminta todella sellaista kuin oli tarkoitus. Fuel Alertin kehityksessä ei ideaalisuuteen päästy, mutta testit kuitenkin kirjoitettiin ainakin osittain yhtä aikaa itse toiminnallisuuden koodaamisen kanssa.

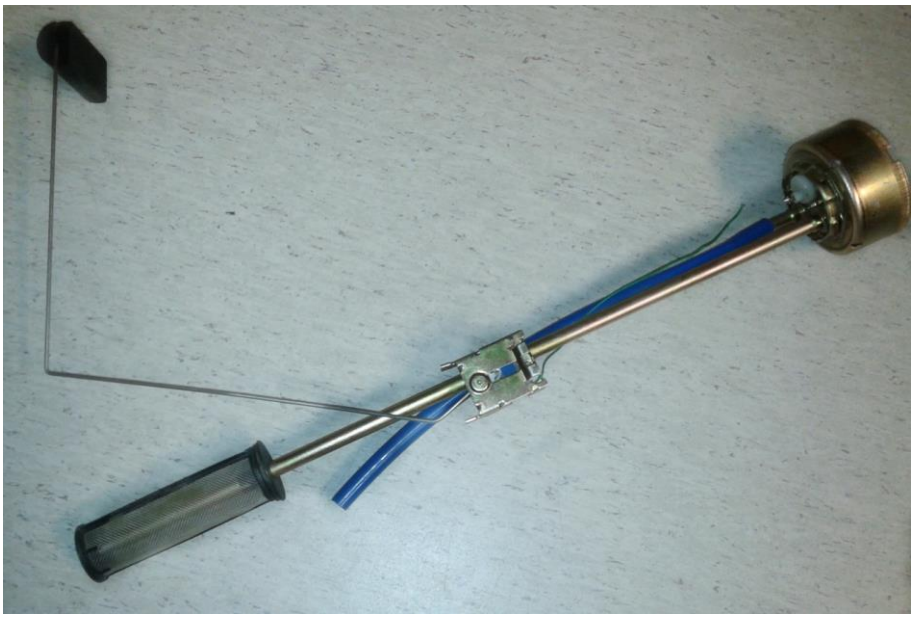
Yksikkötestien tekemiseen käytettiin JUnit-sovelluskehystä. Se on Calgaryn yliopistossa kehitetty ja hyvin laajasti käytetty kehys Java-ohjelmointikielelle.

Yksikkötestien kattavuutta mittaamaan on olemassa monia työkaluja. Tässä tapauksessa käytettiin, lähinnä kokeilumielessä EclEmma Coverage –työkalua, joka on ilmaiseksi la-

dattava lisäosa Eclipseen. Työkalu näyttää graafisesti ja hyvin havainnollisesti sen mitkä rivit koodissa testin aikana on ajettu ja mitkä ei.

5.5.3 Systeemitestaus

Systeemitason testit ovat testejä, joilla pyritään löytämään ongelmat kokonaisuuden toiminnassa. Fuel Alertin tapauksessa testaus tapahtui aluksi pöydällä, johon oli rakennettu testiympäristö polttoaineantureineen. Ajoneuvon liikettä simuloitiin liikuttamalla laitetta käsin ja toimintaa seurattiin debug-lokista. Ohjelmiston kriittisiin kohtiin oli lisätty lokituskomentoja esimerkiksi muuttujien arvoista. Tässä vaiheessa luonnollisesti lokitettavia asioita oli paljon, jotta oli helppo nähdä missä kohdassa mahdolliset ongelmat ilmenivät. Kuviossa 15 on testaukseen käytetty polttoaineanturi.



Kuvio 15. Testauksen alkuvaiheessa käytetty polttoaineanturi

Sekä varsinaisen kehitystyön aikana että sen jälkeen systeemitestaukseen käytettiin myös LabView-ohjelmointikielellä tehtyjä testejä. Testit automatisoimalla toistomääriä saatiin lisättyä huomattavasti ja virheiden toistaminen oli helpompaa, koska testit olivat joka kerta täsmälleen samanlaiset. Se on huomattavasti hankalampi saavuttaa käsin, eikä ole muutenkaan mielekästä heiluttaa polttoaineanturia edestakaisin satoja kertoja peräkkäin käsin kun konekin voi sen tehdä.

Testeillä pyrittiin löytämään virheitä erityisesti hälytyksen lähetyslogiikasta ja selvittämään sopivia raja-arvoja eri parametreille. LabView-ympäristöllä pystyttiin ohjaamaan jännitelähteitä päälle ja pois ja säätämään niiden jännitteitä. Tällä tavalla saatiin simuloitua joitain reaalielämän ajoneuvokäytössä tapahtuvia tapahtumia. Lisäksi rakennetun logiikan avulla pystyttiin lukemaan A1-laitteen debug-lokia, ja näin tutkimaan tekeekö laite sitä mitä siltä odotetaan kaikissa simuloitavissa tilanteissa. Testiohjelmisto laitettiin myös kirjoittamaan raportti, josta näki jokaisen testikierroksen tulokset.

Tärkeä osa testilaitteistoa oli myös A1-laitteen kylkeen kiinnitettävä pienellä sähkömoottorilla toimiva vibraattori. Sillä simuloitiin ajoneuvon liikettä ja sen toimintaa säädettiin LabView:llä ohjattavalla jännitelähteellä.

Testitapauksista tärkeimmät olivat liikkeen aikana tapahtuva polttoainepinnan vaihtelu, pysähdyksissä ollessa tapahtuva polttoainepinnan vaihtelu, sekä testi joka simuloi lyhyttä ajoa, eli jatkuvasti tulee satunnaisin väliajoin virran päälle ja pois kytkemistä. Testien kuvaukset ovat taulukossa 1 ja kytkennät liitteessä 2.

Testin nimi	Tarkoitus	Esiehto	Läpimenoehto
Ignition on/off	Simuloi lyhyttä ajoa, jossa sytytysvirtaa kytketään jatkuvasti päälle ja pois.	A1 päällä, jännitelähteet päällä.	Lokissa näkyy anturin arvon tallennus.
Päävirtakytkin on/off	Simuloi raskaissa ajoneuvoissa olevan päävirtakytkimen käyttöä, joka katkaisee sähköt koko autosta.	A1 päällä, jännitelähteet päällä.	Ei tule hälytystä
Liikettä ja PA-taso vaihtelee	Simuloi ajon aikana tapahtuvaa polttoaineen pinnan vaihtelua.	A1 päällä, jännitelähteet päällä. A1 kiinni vibraattorissa.	Ei tule hälytystä
Ei liikettä ja PA-taso vaihtelee	Simuloi polttoaineen pinnan laskemista ja nousemista ajoneuvon ollessa paikallaan.	A1 päällä, jännitelähteet päällä. A1 kiinni vibraattorissa.	Hälytys tulee kun raja ylittyy, mutta ei ennen sitä.
Yhteys anturiin katkeaa	Simuloi johdon katkeamista tai muuta vikaa, joka aiheuttaa AD-muuntimelle tulevan jänniteen putoamisen nollaan.	A1 päällä, jännitelähteet päällä.	Lokissa virheilmoitus anturin kytkennästä.

Taulukko 1. Fuel Alertin systeemitestit (Fuel Alert testaussuunnitelma 2014.)

5.5.4 **Kenttätestaus**

Kenttätestien suorittaminen oli hieman haasteellista, koska tarkoitukseen sopivaa raskasta kalustoa ei ollut saatavilla vapaaseen käyttöön. Onneksi kuitenkin Aplicomin yhteistyökumppanilta löytyi noin sadan kilometrin päästä kuorma-auto, johon saatiin asennettua A1-laite ja autolle oli myös lähes päivittäisiä ajoja. Koska laite oli sen verran kaukana, piti testauksessa turvautua etätyöskentelyyn. Onneksi A1:n OTAP-toiminto toimii luotettavasti, ja laitteen normaalisti lähettämän datan lisäksi tietyin erityistoimenpitein oli myös debug-loki saatavilla tutkittavaksi. Haastetta aiheutti eniten se, että auto oli normaalissa työkäytössä, ja ajot olivat sen mukaisia. Joskus saattoi mennä montakin päivää, ettei liikehdintää ollut ja tarkkaan kontrolloitujen testien järjestäminen oli muutenkin mahdotonta, koska välitöntä yhteyttä auton käyttäjään ei ollut.

Kenttätesteissä paljastui kuitenkin yksi kohtuullisen merkittäväkin liikkeen tunnistamiseen liittyvä vika, jota ei ollut laboratoriotesteissä havaittu. Liike kyllä tunnistettiin, mutta hieman väärällä tavalla, joka aiheutti Fuel Alertin aktivoitumisen myös kesken ajon. Vika oli jäänyt huomaamatta nimenomaan siksi, että laboratoriotesti oli liian steriili, ja toistokerrat olivat keskenään liian samanlaisia.

5.6 Tuotteistus

Kuten kaikille optioille, myös Fuel Alertille on tuotehallinnan määrittämä tuotekoodi, jolla tuotehallinta yksilöi option ja jota käytetään samalla myös perusteena option aktivoimiseen laitteessa. Option aktivointi on laitekohtaista, eli se on tehtävä erikseen jokaiselle laitteelle jossa optio halutaan käyttöön. Useimmiten ohjelmisto-optio aktivoidaan laitteisiin kun ne lähtevät Aplicomin tuotannosta asiakkaalle, mutta jälkikäteen aktivointi on myös mahdollista.

Ohjelmisto toimitetaan asiakkaille virallisesti Aplicomin extranetin kautta, josta asiakkaat voivat itse ladata viimeisimmän version ohjelmistosta.

5.7 Fuel Alert v2.0

Fuel Alertin kehitys on tapahtunut kahdessa osassa. Ensimmäisen julkaisun jälkeen option käyttöön ottaneilta asiakkailta tuli toive saada valvottua kahden tankin polttoaineenpintaa erikseen. Lisäksi toiseen versioon sisällytettiin jokunen luotettavuutta parantava ominaisuus, ja koodin rakennetta muutettiin modulaarisemmaksi ja helpommin ylläpidettäväksi. Jonkinlaista kehitystä on siis tapahtunut ensimmäisen ja toisen version välillä myös kehittäjän ohjelmointitaidoissa.

Koska ulospäin näkyvät muutokset Fuel Alertin toiminnassa olivat hyvin vähäisiä, myös testaus oli huomattavasti kevyempää kuin ensimmäisen version kanssa. Toimintalogiikka säilyi täysin samana, nyt oli vain kaksi tankkia seurattavana yhtä aikaa. Kuitenkin koodin rakenteen muuttumisen takia oli myös perustoiminnallisuus testattava, ja todettava ettei mikään ollut mennyt rikki.

6 POHDINTA

Työtä aloittaessani minulla oli kokemusta Javasta yhden kurssin verran ja suhteellisen vähän kokemusta ohjelmoinnista yleensäkin. Tuntui siis aika isolta vastuulta kun minulle annettiin tehtäväksi toteuttaa polttoaineen pintaa valvova toiminnallisuus lähtötietojen ollessa hahmotelma kytkennästä ja ohjeet ”jotenkin näin se voisi toimia, tutki mikä olisi paras ratkaisu toteutukseen”.

Ei se kuitenkaan missään vaiheessa tuntunut liian isolta vastuulta, ja vaikka alussa työ eteni hitaasti, niin se ymmärrettiin. Toki loppua kohti tahdin piti kiristyä ja loppuvuoteen 2013 mennessä option ensimmäisen version tuli olla valmis asiakaskäyttöön, jolloin se julkaistiin yhdessä muiden vuoden viimeisessä julkaisussa tulevien ominaisuuksien kanssa.

Luonnollisesti aikaa meni alussa todella paljon perusasioiden opetteluun, en ollut aikaisemmin käyttänyt esimerkiksi Eclipse-kehitystyökalua, enkä kuullutkaan SVN-versionhallinnasta, puhumattakaan Aplicomin laitteiden toiminnasta ja ohjelmistoista.

Yksi uusi ja ennen kokematon asia ohjelmistokehitysprosessiin liittyen olivat katselmoinnit. Sain etukäteen ohjeen varautua siihen, että ”se ei ole mikään itsetunnon nostatustilaisuus”, ja se vähän jännitti ensimmäisellä kerralla. Fuel Alertin dokumenttikatselmointi oli ensimmäinen katselmointi koskaan jossa olin mukana, ja heti katselmoitiin minun kirjoittamaa dokumenttia. Katselmoimassa olivat Aplicomin tekninen johtaja, tuotekehityspäällikkö, huoltopäällikkö sekä johtava ohjelmistosuunnittelija, joten ihan mitätön tilaisuus se ei pienelle harjoittelijalle ollut. Alun pikku jännityksen jälkeen onneksi – ja niin kuin olettaa saattaa – kaikki sujui kuitenkin hyvin. Toki muutoksia vaativia asioita dokumentista löytyi, mutta itsetuntoni ei kyllä kärsinyt, oikeastaan se vain nousi.

Myös testauksesta opin paljon uutta Fuel Alertin kehityksen yhteydessä. Ehkä tärkein testaukseen liittyvä asia jonka opin, oli niinkin perustavaa laatua oleva asia kuin se, että mikä on testaamisen tarkoitus. Luulin että se olisi varmistaa toiminnan oikeellisuus, mutta todellisuudessa se onkin yrittää löytää virheitä – ja ne ovat kaksi eri asiaa. Testauksen

aikana opettelin lisäksi minulle ennestään tuntemattoman LabView-ohjelmointikielen, jolla tein testauksen automatisoinnin.

Omasta mielestäni työ onnistui kaiken kaikkiaan hyvin, tähän päivään mennessä minun korviini ei ole kuulunut negatiivista asiakaspalautetta. Varsinkin toisen version jälkeen minustakin tuntui, että Fuel Alert on nyt oikeasti niin hyvä, että voin hyvillä mielin ajatella sen lähtevän oikeaan asiakaskäyttöön. Kehitys jatkunee edelleen asiakastarpeen mukaan, ja kenties tulevaisuudessa nähdään vieläkin pätevämpi Fuel Alert polttoainetankkeja turvaamassa.

Kokonaisuudessaan työ on ollut erittäin mielenkiintoista ja aivan uskomattoman opettavaista. Ohjelmiston suunnittelu sulautettuun järjestelmään on juuri sitä mitä haluan tehdä ja oppia siitä niin paljon kuin mahdollista. Kun siihen vielä saadaan liitettyä edes jonkin verran elektroniikkaa ja mekaniikkaa, kuten Fuel Alertin tapauksessa, niin minun mielenkiintoni asiaan on taattu.

Haluankin kiittää Aplicom Oy:tä tästä mahdollisuudesta, ja erityisesti tuotekehityspäällikkö Atri Vainikaista ja CTO Juha Savolaista heidän minua kohtaan osoittamastaan luottamuksesta. Toivon, että tämä työ on sen luottamuksen arvoinen.

LÄHTEET

A1 Installation Guide. 2014. Rajoitettu PDF-dokumentti, Aplicom Oy.

A1 Track SW and Telematics SW User Manual. 2014. Rajoitettu PDF-dokumentti, Aplicom Oy.

Aplicom A1 with ready made software. 2009. PowerPoint-esitys, Aplicom Oy.

Connected Limited Device Configuration. 2014. Verkkosivu, Wikipedia - vapaa tietosanakirja. Viitattu 3.8.2014. http://en.wikipedia.org/wiki/CLDC

Fleet Management System. 2014. Verkkosivu, Wikipedia - vapaa tietosanakirja. Viitattu 30.7.2014. http://en.wikipedia.org/wiki/Fleet_Management_System

Fuel Alert SW Option. 2014. Rajoitettu PDF-dokumentti, Aplicom Oy.

Fuel Alert Testaussuunnitelma. 2014. Rajoitettu PDF-dokumentti, Aplicom Oy.

Haikala, I & Märijärvi, J. 2000. Ohjelmistotuotanto. 7. p., uud. p. Helsinki: Talentum Media Oy.

Java ME. 2014. Verkkosivu, Wikipedia - vapaa tietosanakirja. Viitattu 3.8.2014. http://fi.wikipedia.org/wiki/Java_ME

Java Platform, Micro Edition. 2014. Verkkosivu, Wikipedia - vapaa tietosanakirja. Viitattu 3.8.2014. http://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition

Java Virtual Machine Specification. 2014. Verkkojulkaisu, Oracle. Viitattu 10.9.2014. http://docs.oracle.com/javase/specs/jvms/se7/html/index.html

Savolainen, J. 2014. Tekninen johtaja. Aplicom Oy. Haastattelu 12.8.2014.

TC65i-X Java User's Guide. 2012. Rajoitettu PDF-dokumentti, Gemalto Gmbh.

Telematics. 2014. Verkkosivu, Wikipedia - vapaa tietosanakirja 2014. Viitattu 20.6.2014. http://en.wikipedia.org/wiki/Telematics

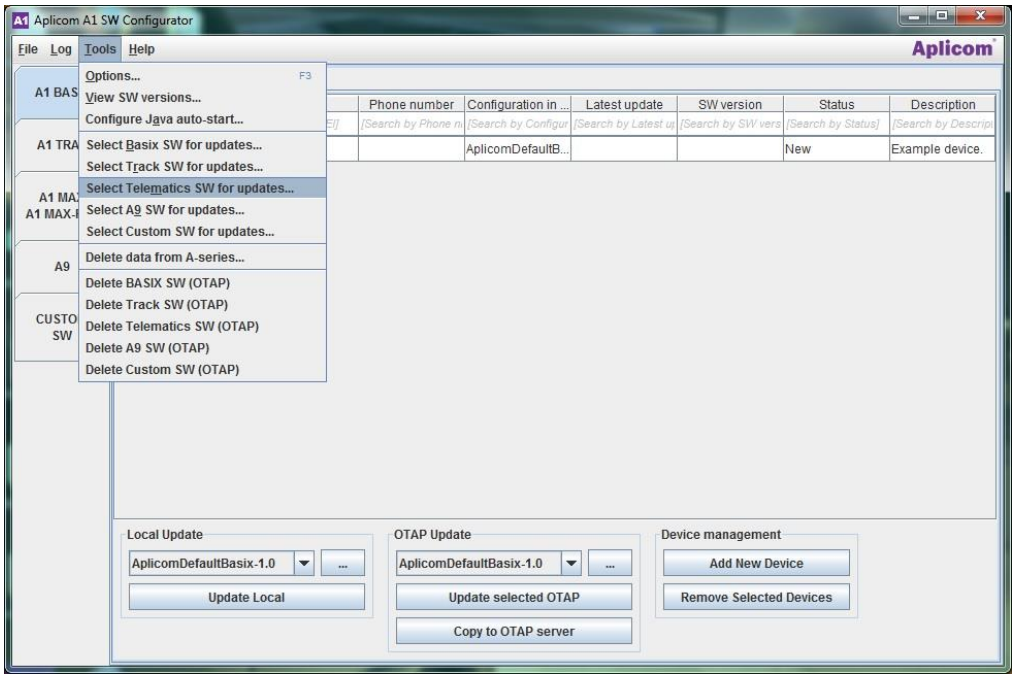
The K Virtual Machine. 2014. Verkkosivu, Oracle. Viitattu 3.8.2014 http://www.oracle.com/technetwork/java/ds-137153.html

Toimintajärjestelmän kuvaus. 2014. PDF-dokumentti, Aplicom Oy.

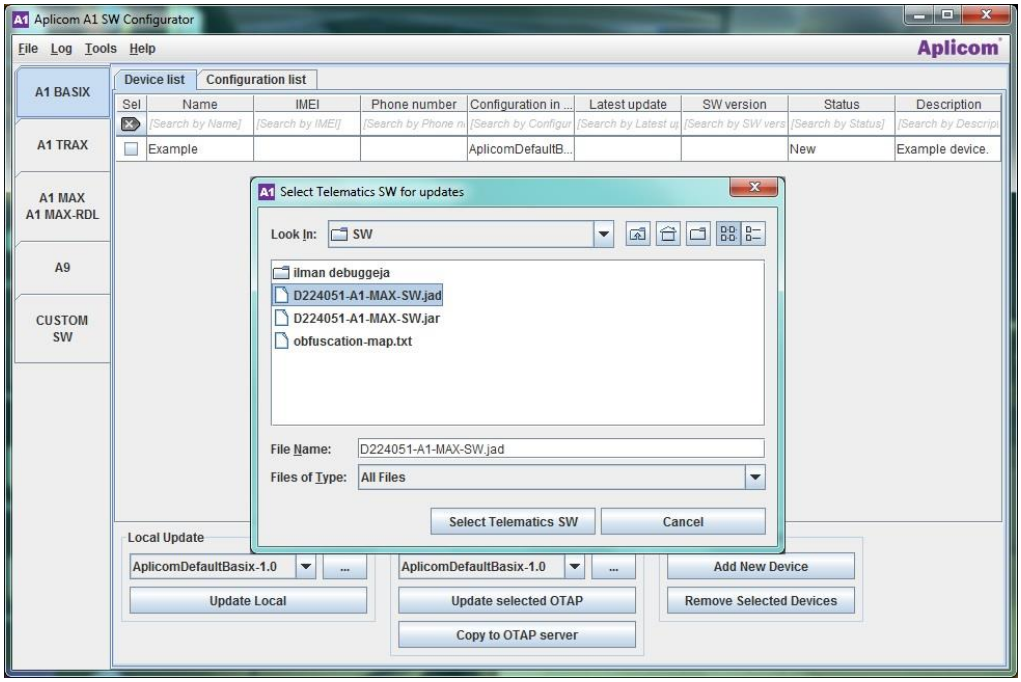
LIITTEET

Liite 1, Ohjelmiston lataaminen A1-laitteeseen A1 SW Configuratorilla

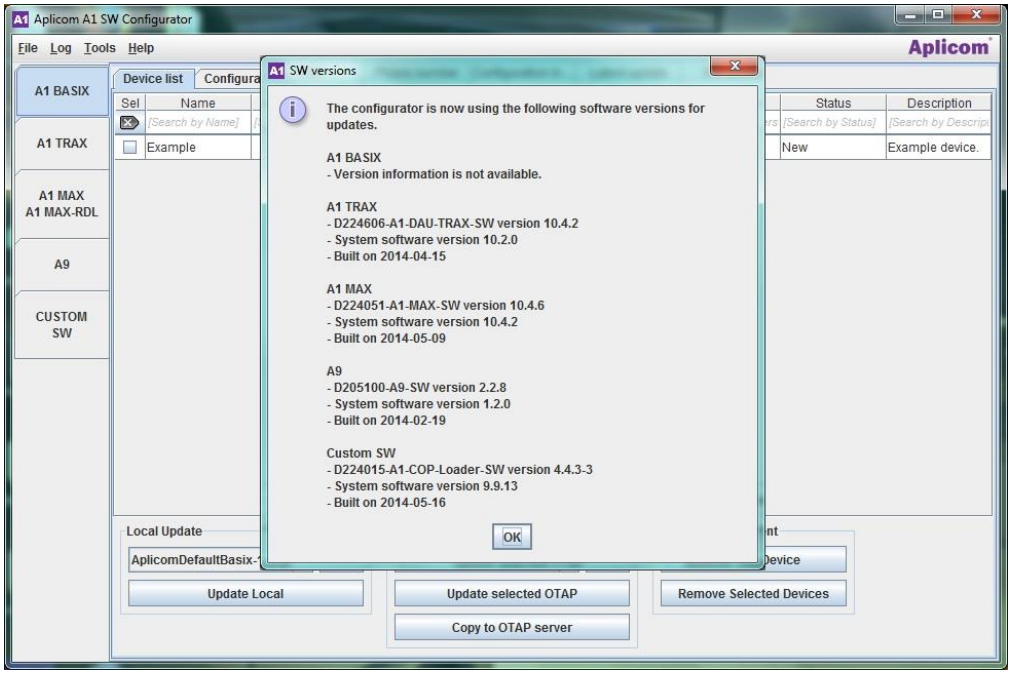
Kun A1-laite on päällä ja kaapelit kytketty, pitää A1 SW Configuratorin asetuksista valita oikea COM-portti käyttöön, sekä oikea ohjelmisto ja konfiguraatiotiedosto laitteeseen ladattavaksi.



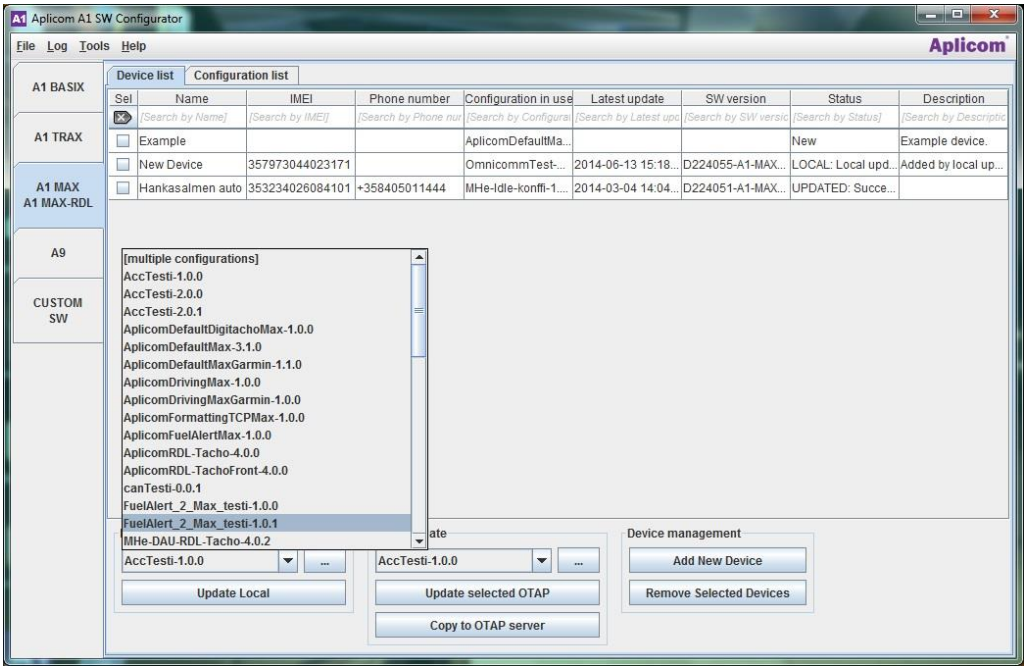
Kuvio 1. Ladattavan ohjelmiston valinta A1 SW Configuratorissa



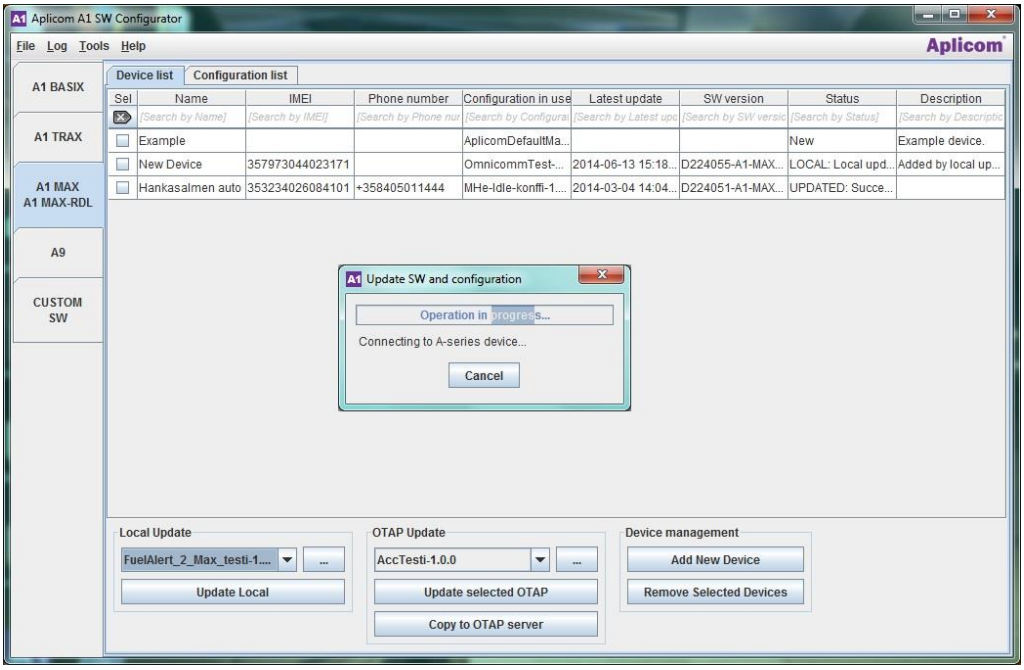
Kuvio 2. Ladattavan ohjelmiston valinta A1 SW Configuratorissa



Kuvio 3. Configurator näyttää kullekin laitteelle päivitystä varten valitut ohjelmistot

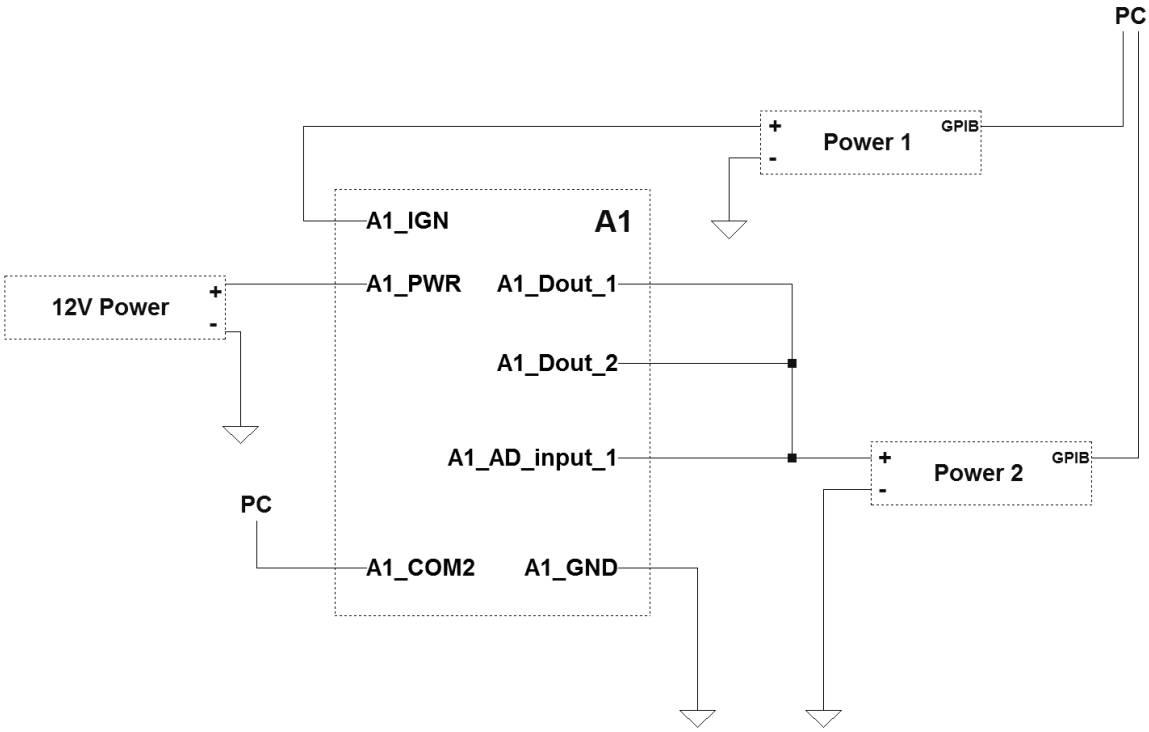


Kuvio 4. Listalta valitaan haluttu konfiguraatiotiedosto ladattavaksi laitteeseen

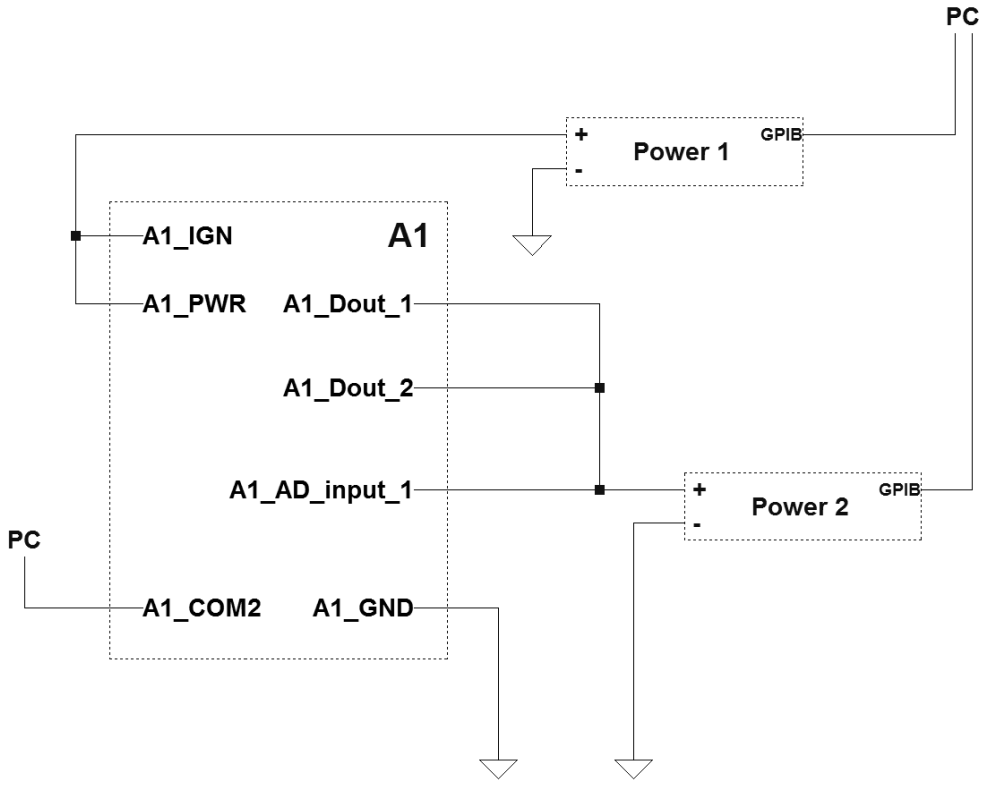


Kuvio 5. Painamalla ”Update Local”-painiketta vasemmassa alareunassa Configurator aloittaa laitteen ohjelmiston ja konfiguraation päivityksen

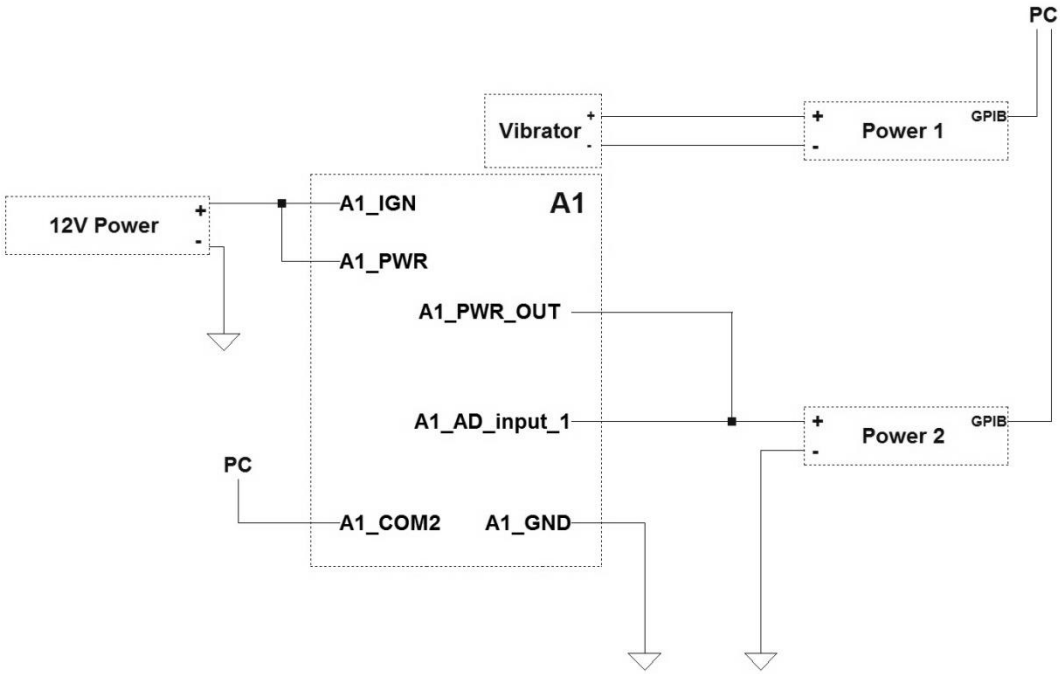
Liite 2, Testien kytkennät



Kuvio 1. Ignition on/off –testin kytkentä.



Kuvio 2. Päävirtakytkin on/off -testin kytkentä



Kuvio 3. Liikettä vaativien testien kytkentä